Project Excelsior

Abhishek Dutta
abhishek.dutta1337@gmail.com

Table of contents

Table of figures

Introduction

Excelsior is an online comic book retailer, and a store for comic fans that operates in a way like its competitor Mile High Comics [3]. The domain of the project is e-commerce, and the intended application is to support the front-end business application by providing the backend database design and enabling high-performance querying for enquiries by the users.

Our vision is to create a scalable, highly effective database system that enables us to offer our consumers a seamless and satisfying online experience. The success of our company depends on having a solid database architecture in the fiercely competitive market of online comic book sales. Our intended application is to create a database system that can efficiently store and retrieve customer data, order information, and inventory details.

By ensuring the reliability and scalability of our database system, we can continue to compete effectively with other online comic book retailers like Mile High Comics.

The database design would aim to cover most of the generic enquiries that a user may put through the front-end application. These would support basic enquiries such searching for comic books by title, publisher, year, price, and quality. It would also support advanced enquiries based on a combination of the above-mentioned factors for a more specific requirement of results.

The underlying data would be frequently changing as new comic book issues come out, as well as existing issues would get updated in the database. In terms of scale, there would be a very large number of records in the underlying database tables, since each comic book series would be accompanied by its various issues along with the basic attributes such as price, quality, year etc.

Database Plan

For an efficient database plan, it is essential to consider the end-user requirements. The main objective is to offer a safe, scalable, and efficient database. Customer information, product details, and order information will all be stored in the database. To assure consistency and integrity of the data, the database design will adhere to best practices for normalization and will work to reduce data redundancy. The database and the business's e-commerce platform will be connected, allowing for seamless data transfer between the database and the front-end application.

Based on the requirements for Excelsior, the set of entities and their attributes are as follows:

1. Customers

Customers represent the entities which would have often have a transactional relationship with our database, meaning that they would often be connected to orders. Its main attributes are as follows:

1. Unique identifier for a customer
2. Full name of the customer
3. E-mail address of the customer
4. Phone number of the customer
5. City of residence of the customer

2. Orders

The orders represent all the orders that were placed by customers while they made the comic book purchases. Its main attributes are as follows:

1. Unique identifier for a customer
2. The unique customer identifier to which the order belongs to
3. Unique product identifier which belongs to the order
4. Date of order placed.

5. Most recent status of the order

3. Inventory

The inventory represents all the unique products available in Excelsior and their quantity. Its main attributes are as follows:

1. Unique identifier of the product
2. Quantity of the product available

4. Comic books

The comic book represents the main product that is sold via Excelsior. Its main attributes are as follows:

1. Unique product identifier which represents each comic book
2. The title of the comic book
3. The issue number of the comic book
4. Unique identifier for the condition of the comic book
5. Price of the comic book
6. Genre of the comic book
7. The name of the publisher of the comic book
8. Unique identifier for the writer of the comic book
9. Year in which the common book was written.
10. Unique identifier of the first main character of the comic book
11. Unique identifier of the second main character of the comic book

5. Writers

The writer represents the author of the comic book. Its main entities are as follows:

1. Unique identifier for the comic book writer
2. The name of the comic book writer
3. Unique identifier for the character that the writer has created the comic book about.

6. Comic characters

A character represents a fictional entity which may exist in multiple comic books. Its main attributes are:

1. Unique identifier for the character
2. Name of the character

7. Comic condition

Comic condition represents a lookup for the condition of the comic books that are available in the database. These entities have main attributes as:

1. Unique identifier for comic condition
2. The description of the condition
3. A numeric value representing the condition itself (ranges from 0 to 10)

Entity-Relationship (ER) diagram

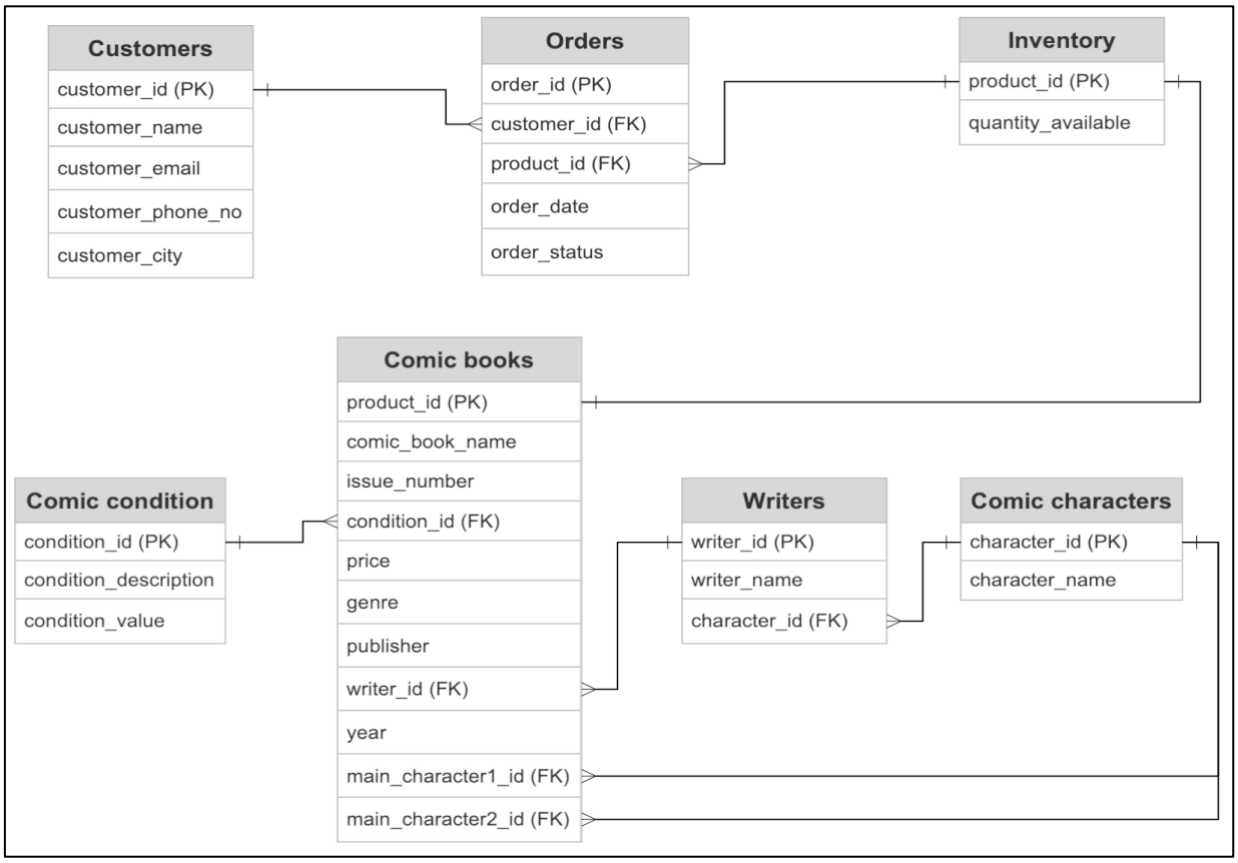


Figure 1. ER diagram for Excelsior

The above ER diagram was built using https://www.smartdraw.com

The main relationships between the entities and the rationale behind them, are described follows:

1. The customer entity is connected to the orders entity via the unique customer identifier – this is done so that each order may be linked to the customer who placed the order.
2. The orders entity is connected to the inventory entity via the unique product identifier – it is so because the inventory entity would contain the list of all unique products that Excelsior has to offer.
3. The inventory entity is connected to the comic book entity via the unique product identifier – it is so because we would want to track the quantity available of each of our products.
4. The condition entity is connected to the comic book entity via the unique condition identifier – it is so because the condition entity represents a mapping between symbolic and numerical quality codes.
5. The comic book entity is connected to the writer entity via the unique writer identifier – it is so because if required then the power-user may be able to lookup the writer of the comic book as well as look for other characters that the writer may have written.
6. The writer entity is connected to the character entity via the unique character identifier – it is so because a power user may want to look for all the different characters that the writer has written.

Additional database plan rationale

1. The writers and character entities are segregated from the comic book entity because most of the user's requirements for comic book search can be fulfilled just by using the comic book entity. In case they want a refined search based on the writers or characters, they may choose to do so, but this would be relatively less frequent as compared to a simple comic book search. This segregation would also help in query performance because most of the basic

search through the comic books can be done without joining with either the writer or character entities.

2. The quantity available attribute of a comic book is stored in the inventory entity and not the comic book entity itself, because it is more of an operational attribute meaning that the power-user may not always essentially care about the numeric quantity available for a comic book in the backend if they can just see that it is available. The quantity available can be used by other teams in the organization for stock planning.

Database Structure

The database will contain the following seven tables representing the entities defined in the previous section.

1. customers

The customers table would contain the details for the customers that have purchased a comic book from Excelsior. Each row in the table represents a unique customer. The details of the table are as follows:

Column name	Column data type	Column constraints
customer_id	bigint	Primary key
customer_name	varchar(50)	-
customer_email	varchar(100)	-
customer_phone	varchar(14)	-
customer_city	varchar(20)	-

Figure 2. customer table's structure

In the customers table, the primary key is customer_id, which is a unique identifier for each customer.

The functional dependencies in the table are as follows:

customer_id → customer_name, customer_id → customer_email, customer_id → customer_phone , customer_id → customer_city

In this case, each functional dependency has customer_id as the determinant, which is a candidate key and therefore a superkey. Therefore, the customers table is in BCNF. Since the table is in BCNF, it is also in 1NF, 2NF and 3NF. [1]

2. orders

The orders table would contain the order details for all orders that have been placed through Excelsior. Each row in the table represents a unique order placed. The details of the table are as follows:

Column name	Column data type	Column constraints
order_id	bigint	Primary key
customer_id	bigint	Foreign key
product_id	bigint	Foreign key
order_date	datetime	-
order_status	varchar(15)	-

Figure 3. orders table's structure

In the orders table, the primary key is order_id, which is a unique identifier for each order. There are also two foreign keys, customer_id and product_id, which reference the customers and products tables respectively.

The functional dependencies in the table are as follows: order_id → customer_id, order_id → product_id, order_id → order_date, order_id → order_status

In this case, each functional dependency has order_id as the determinant, which is a candidate key and therefore a superkey. Therefore, the orders table is in BCNF. Since the table is in BCNF, it is also in 1NF, 2NF and 3NF.

3. inventory

The inventory table represents all the products that are available in Excelsior. Each record in the table represents a single product along with its quantity. The details of the table are as follows:

Column name	Column data type	Column constraints
product_id	bigint	Primary key
quantity_available	bigint	-

Figure 4. inventory table's structure

The functional dependency in the table is: product_id → quantity_available
In this case, the only functional dependency has product_id as the determinant, which is a candidate key and therefore a superkey. Therefore, the inventory table is in BCNF. Since the table is in BCNF, it is also in 1NF, 2NF and 3NF.

4. comic_characters

The comic_characters table represents the set of all characters that are available in the Excelsior database for comic books. Each record in the table refers to each comic book character. The details of the table are as follows:

Column name	Column data type	Column constraints
character_id	bigint	Primary key
character_name	varchar(255)	-

Figure 5. comic_characte table's structure

There is only one functional dependency: character_id → character_name
In this case, the only functional dependency has character_id as the determinant, which is a candidate key and therefore a superkey. Therefore, the comic_characters table is in BCNF. Since the table is in BCNF, it is also in 1NF, 2NF and 3NF.

5. writers

The writers table represents all the writers of different characters available in the Excelsior database. Each record represents a unique writer. A writer may have written multiple comic book characters. The details of the table are as follows:

Column name	Column data type	Column constraints
writer_id	bigint	Primary key
writer_name	varchar(50)	-
character_id	bigint	Foreign key

Figure 6. writers table's structure

6. comic_book_details

The comic_book_details is the core table in our database which would be the most widely used by different kinds of users. It contains basic information about each comic book available as part of the inventory. Each record represents a unique comic book. The details are as follows:

Column name	Column data type	Column constraints
product_id	bigint	Primary key
comic_book_name	varchar(255)	-
issue_number	bigint	-
condition_id	bigint	Foreign key
price	bigint	-
genre	varchar(25)	-
publisher	varchar(100)	-
writer_id	bigint	Foreign key
year	bigint	-
main_characterid_1	bigint	Foreign key
main_characterid_2	bigint	Foreign key

Figure 7. comic_book_details table's structure

In this table, we have functional dependencies as follows:

1. product_id -> comic_book_name, issue_number, condition_id, price, genre, publisher, writer_id, year, main_characterid_1, main_characterid_2
2. writer_id -> writer_name
3. condition_id -> condition_description
4. main_characterid_1 -> character_name
5. main_characterid_2 -> character_name

In our case, all the functional dependencies are dependent on the primary key "product_id," which is a candidate key. Therefore, the table "comic_book_details" is in BCNF. Since the table is in BCNF, it is also in 1NF, 2NF and 3NF.

7. comic_condition

The comic_condition table is a lookup table which describes the comic book conditions. Every record in the table represents a unique comic book condition value along with its description. The details of the table are as follows:

Column name	Column data type	Column constraints
condition_id	bigint	Primary key
condition_description	varchar(15)	-
condition_value	decimal(3,1)	-

The only candidate key is the primary key (condition_id) and there are no non-trivial functional dependencies. Therefore, the table is in BCNF form. Since the table is in BCNF, it is also in 1NF, 2NF and 3NF.

Database Views

This section provides a few examples of the different kinds of database views that can be built using the database tables as described in the previous section. The views may be utilized by Excelsior staff for stock planning, as well as by end users who wish to view the catalogue of comic book offerings based on different filters.

1. top_customers view

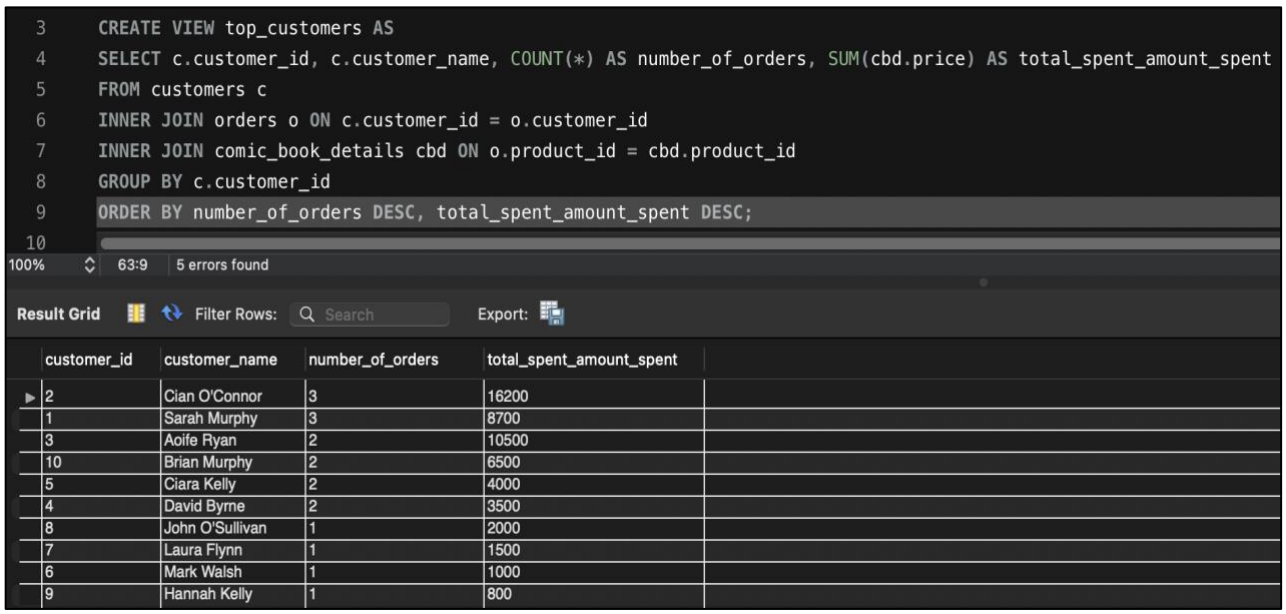
This view would provide a list of all customers based on the descending order of the number of orders placed by them and the total amount of money spent by them while making purchases from Excelsior. This is created as a view and not a table because if it were a table, it would need to be refreshed regularly.

Target audience: This view is supposed to be used by the internal Excelsior team for data anaysis purposes, and would help the marketing team as well for making decisions.

The view would return the following columns:

1. customer_id – taken from customers table
2. customer_name – taken from customers table
3. total number of orders placed – calculated column
4. total amount spent – calculated column

The view definition and sample output is as follows:



```
3   CREATE VIEW top_customers AS
4   SELECT c.customer_id, c.customer_name, COUNT(*) AS number_of_orders, SUM(cbd.price) AS total_spent_amount_spent
5   FROM customers c
6   INNER JOIN orders o ON c.customer_id = o.customer_id
7   INNER JOIN comic_book_details cbd ON o.product_id = cbd.product_id
8   GROUP BY c.customer_id
9   ORDER BY number_of_orders DESC, total_spent_amount_spent DESC;
10
```

100% 63:9 5 errors found

Result Grid Filter Rows: Search Export:

customer_id	customer_name	number_of_orders	total_spent_amount_spent
2	Cian O'Connor	3	16200
1	Sarah Murphy	3	8700
3	Aoife Ryan	2	10500
10	Brian Murphy	2	6500
5	Ciara Kelly	2	4000
4	David Byrne	2	3500
8	John O'Sullivan	1	2000
7	Laura Flynn	1	1500
6	Mark Walsh	1	1000
9	Hannah Kelly	1	800

Figure 8. top_customers view

2. consolidated_comic_books view

This view would be a go-to view for searching through the catalogue of comic books available on Excelsior. It would contain all the required filters as columns on which the search may provide their search query.

Target audience: This view would be majorly used by the users or customers who intend to browse through the comic book catalogue. This view would be integrated with the main front-end application which would enable a user-friendly method of interacting with the view.

The view would return the following columns:

1. product_id – taken from comic book details
2. comic_book_name – taken from comic book details
3. comic condition description – taken from comic_conditon table
4. comic_condition value – taken from comic_condition table
5. price - taken from comic book details
6. genre - taken from comic book details
7. publisher - taken from comic book details
8. main_character1_name - taken from comic characters
9. main_character2_name – taken from comic characters
10. writer_name – taken from writers table
11. year – taken from comic book details

The view definition and sample output is as follows:

```
CREATE VIEW consolidated_comic_books AS
select cbd.product_id, cbd.comic_book_name,
cc.condition_description,
cc.condition_value,
cbd.price,
cbd.genre,
cbd.publisher,
cc1.character_name as main_character1,
cc2.character_name as main_character2,
w.writer_name,
cbd.year
 from comic_book_details cbd
 left join comic_condition cc
 on cbd.condition_id = cc.condition_id
 left join writers w
 on cbd.writer_id = w.writer_id
 left join comic_characters cc1
 on cbd.main_character1_id = cc1.character_id
 left join comic_characters cc2
 on cbd.main_character2_id = cc2.character_id;
```

100% 41:33 5 errors found

Result Grid Filter Rows: Search Export:

product_...	comic_book_name	condition_descript...	condition_val...	price	genre	publisher	main_character1	main_characte...	writer_name	year
1001	Batman: Year One	Mint(MT)	10.0	2500	Superhero	DC Comics	Batman	Batman	John Smith	1987
1002	The Amazing Spider-Man	Near-Mint(NM)	9.5	5000	Superhero	Marvel Comics	Spider-Man	Spider-Man	Jane Doe	1988
1003	Watchmen	Very Fine(VF)	8.0	10000	Superhero	DC Comics	Rorschach	Rorschach	Bob Johnson	1986
1004	The Walking Dead	Fine(FN)	6.0	500	Horror	Image Comics	Rick Grimes	Rick Grimes	Sara Lee	2003
1005	Sandman	Mint(MT)	10.0	3000	Fantasy	DC Comics	Dream	Dream	Mike Brown	1989
1006	Spawn	Very Fine(VF)	8.0	1000	Superhero	Image Comics	Spawn	Spawn	Karen Davis	1992

Figure 9. consolidated_comic_books view

The value addition by this view is that it is devoid of any identifier columns(except product id) and only contains those columns which can be used as filters during comic book search in the front-end application. Hence, it is expected to be performant.

3. stock_check view

The stock_check view would act as a go-to place for checking the stocks of different comic books based on their genre, publisher as well as character name.
The view would return the following fields:

1. genre – taken from comic book details table
2. publisher – taken from comic book details table
3. character name – taken from comic_characters table
4. total quantity available – taken from inventory table

Target audience: This view would primarily used by internal Excelsior team members, especially from the operations and logistics team. The view would help them plan their stock according to the requirement based on genre, publisher or comic character name.

```
52 •  CREATE VIEW stock_check AS
53    select
54    cbd.genre, cbd.publisher, cc.character_name, sum(i.quantity_available) as total_quantity_available
55     from comic_book_details cbd
56    left join comic_characters cc on cc.character_id IN (cbd.main_character1_id, cbd.main_character2_id)
57    left join inventory i on cbd.product_id = i.product_id
58    group by 1,2,3;
59
```

00% 16:58 3 errors found

Result Grid Filter Rows: Search Export:

genre	publisher	character_name	total_quantity_availa...
Superhero	DC Comics	Batman	25
Superhero	Marvel Comics	Spider-Man	10
Superhero	DC Comics	Rorschach	50
Horror	Image Comics	Rick Grimes	30
Fantasy	DC Comics	Dream	20
Superhero	Image Comics	Spawn	40
Horror	Dark Horse Comics	Hellboy	15
Crime	Dark Horse Comics	Marv	35
Superhero	Vertigo	Jesse Custer	45
Science Fiction	Image Comics	Alana	5
Historical	Pantheon Books	Art Spiegelman	5
Science Fiction	Vertigo	Yorick Brown	10

Figure 10. stock_check view

The value add of this view is that it simplifies the stock planning process for the company. Users may also drill-up/group-by on a particular column such as genre, publisher or character name as well as filter for a particular value of the same. As the database grows larger, this view would only become more valuable.

4. ongoing_orders view

This view would provide the details for all non-delivered ongoing orders along with their respective customer details.

The view would return the following fields:

1. customer_id – taken from customers table
2. customer_name – taken from customers table
3. customer_email – taken from customers table
4. customer_phone_number – taken from customers table
5. customer_city – taken from customers table
6. order_date – taken from orders table
7. order_status – taken from orders table
8. order_id – taken from orders table

Target audience: This view would primarily used by the operations team to determine which orders need more attention/expedition.

```
42  CREATE VIEW ongoing_orders AS
43  select c.*, o.order_date, o.order_status, o.order_id from orders o
44  left join customers c on o.customer_id = c.customer_id
45  where order_status <> 'delivered';
```

100% 35:45

Result Grid Filter Rows: Search Export:

customer_id	customer_name	customer_email	customer_phone_...	customer_ci...	order_date	order_status	order_id
1	Sarah Murphy	sarah.murphy@gmail.com	01-555-9876	Dublin	2023-05-03 10:00:00	pending	1
2	Cian O'Connor	cian.oconnor@gmail.com	086-555-4321	Cork	2023-04-27 11:00:00	shipped	2
4	David Byrne	david.byrne@gmail.com	085-555-2222	Galway	2023-05-03 13:00:00	pending	4
5	Ciara Kelly	ciara.kelly@gmail.com	087-555-1111	Dublin	2023-05-03 14:00:00	shipped	5
7	Laura Flynn	laura.flynn@gmail.com	086-555-7777	Dublin	2023-05-03 16:00:00	pending	7
8	John O'Sullivan	john.osullivan@gmail.com	087-555-3333	Cork	2023-05-01 17:00:00	shipped	8
10	Brian Murphy	brian.murphy@gmail.com	087-555-4444	Limerick	2023-05-03 19:00:00	pending	10
1	Sarah Murphy	sarah.murphy@gmail.com	01-555-9876	Dublin	2023-04-20 20:00:00	shipped	11
3	Aoife Ryan	aoife.ryan@gmail.com	087-123-4567	Limerick	2023-05-03 22:00:00	pending	13
4	David Byrne	david.byrne@gmail.com	085-555-2222	Galway	2023-04-17 23:00:00	shipped	14

Figure 11. ongoing_orders view

Procedural elements

This section covers the procedural elements that will be incorporated into the Excelsior database. These include triggers and stored procedures which can be called as required.

Note: *Instead of PL/SQL, the database would use SQL/PSM since Excelsior's database is based on MySQL.*

1. Remove abandoned orders procedure

This procedure would remove all the orders from the orders table which have an order date of later than 1 month from the current date and are still in the 'pending' status. This would help us remove stale orders from the table as well help in maintaining the ever-growing size of the table in terms of total number of records as well as the space taken.

The definition and invokation of the procedure is as follows:

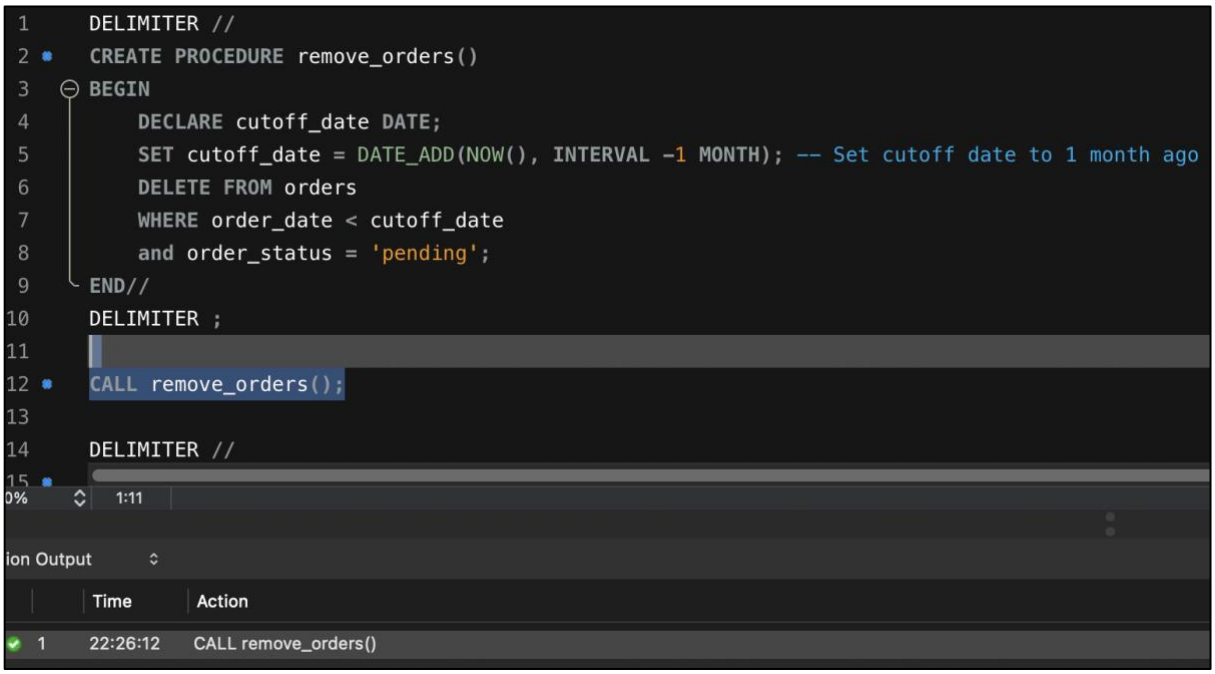


Figure 12. Procedure for removing abandoned orders

2. Apply discount procedure

The apply discount procedure would apply a user-given percentage of discount based on a user-given month of the year, to all comic books. It provides a flexibility of applying a discount at once, based on a condition. The procedure may be modified to incorporate multiple conditions. For example, we could change the inner condition and say that the discount be applicable to a specific genre of comic books, or to comic books which belong to a particular publisher.

The definition and invokation of the procedure is as follows:

```
DELIMITER //
CREATE PROCEDURE apply_discount(discount_percentage decimal(3,1), month_number bigint)
BEGIN
UPDATE comic_book_details cbd
JOIN
(
    select product_id, price, price*(1-discount_percentage) as discounted from comic_book_details
    WHERE MONTH(NOW()) = month_number
    AND YEAR(NOW()) = 2023
    group by 1,2,3
)ref ON cbd.product_id = ref.product_id
set cbd.price = ref.discounted;
END//
DELIMITER ;
```

Figure 13. apply discount procedure

```
select * from comic_book_details;

-- Apply 20% discount for the month of May in 2023
CALL apply_discount(0.2, 5);
```

34:29

Result Grid | Filter Rows: Search | Export:

product_...	comic_book_name	issue_number	condition...	price	genre	publisher	writer_id	year	main_character1_id	main_character2...
1001	Batman: Year One	1	1	2500	Superhero	DC Comics	1	1987	1	1
1002	The Amazing Spider-Man	300	2	5000	Superhero	Marvel Comics	2	1988	2	2
1003	Watchmen	1	3	10000	Superhero	DC Comics	3	1986	3	3

Before

```
-- Apply 20% discount for the month of May in 2023
CALL apply_discount(0.2, 5);

select * from comic_book_details;
```

34:34

Result Grid | Filter Rows: Search | Export:

product_...	comic_book_name	issue_number	condition...	price	genre	publisher	writer_id	year	main_character1_id	main_character2...
1001	Batman: Year One	1	1	2000	Superhero	DC Comics	1	1987	1	1
1002	The Amazing Spider-Man	300	2	4000	Superhero	Marvel Comics	2	1988	2	2
1003	Watchmen	1	3	8000	Superhero	DC Comics	3	1986	3	3

After

As visible from the above screenshots, after running the procedure, the price of all comic books during the month of May 2023 was reduced by 20%

3. Update inventory trigger

This trigger [2] would be executed after a new record is inserted into the orders table. The product id contained within the new order, would be matched with the inventory table, and the corresponding quantity available for that product, would be reduced by one. Hence, updating the inventory table automatically.

The definition of the procedure is as follows:

```
36  DELIMITER //
37  CREATE TRIGGER decrease_quantity_available
38  AFTER INSERT ON orders
39  FOR EACH ROW
40  BEGIN
41      UPDATE inventory
42      SET quantity_available = quantity_available - 1
43      WHERE product_id = NEW.product_id;
44  END//
45  DELIMITER ;
46
47
100%   1:47
```

Figure 14. update inventory trigger

4. Disallow inserting new comic conditions

This trigger acts as a blocker for any query which tries to insert a new record into the comic conditions table. This ensures that there are always the same number of records in the comic conditions table. It will raise an error if any query tries to insert a new record into the lookup table.

The trigger definition and sample output is as follows:

```
DELIMITER //
CREATE TRIGGER trg_insert_condition_lkp
BEFORE INSERT ON comic_condition
FOR EACH ROW
BEGIN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insertions into condition_lkp table are not allowed';
END//
DELIMITER ;

INSERT INTO `comic_condition` (condition_id, condition_description, condition_value)
VALUES
  (50, 'So Good(SG)', 0.1);

-- Error Code: 1644. Insertions into condition_lkp table are not allowed
```

1:63 1 error found

Action Output

	Time	Action	Response
1	00:08:07	drop trigger trg_insert_condition_lkp	0 row(s) affected
2	00:08:11	CREATE TRIGGER trg_insert_condition_lkp BEFORE INSERT ON comic_condition FOR EACH ROW BEGIN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Inser...	0 row(s) affected
3	00:08:16	INSERT INTO `comic_condition` (condition_id, condition_description, condition_value) VALUES (50, 'So Good(SG)', 0.1)	Error Code: 1644. Insertions into condition_lkp table...

Figure 15. disallow insert into comic conditions trigger

Example queries

Below are a few example queries that are tested against the Excelsior tables and views created so far. These would cover some of the major use-cases for different kinds of users.

1. Searching for comic books based on multiple conditions

```
select * from consolidated_comic_books
where condition_value > 8
and price < 7000
and genre = 'Superhero';
```

100% 25:4

Result Grid Filter Rows: Search Export:

product_...	comic_book_name	condition_descript...	condition_val...	price	genre	publisher	main_character1	main_characte...	writer_name	year
1001	Batman: Year One	Mint(MT)	10.0	2500	Superhero	DC Comics	Batman	Batman	John Smith	1987
1002	The Amazing Spider-Man	Near-Mint(NM)	9.5	5000	Superhero	Marvel Comics	Spider-Man	Spider-Man	Jane Doe	1988

Figure 16. multiple condition search query

In the above query, the end-user tries to look for all comic books which have a condition value of more than 8, a price which is less than 7000 and genre as

‘Superhero’. It demonstrates that the underlying view is capable of basic search queries for comic books.

2. Examining ongoing orders for customers which belong to a particular city

```
6  select * from ongoing_orders
7  where order_status = 'shipped'
8  and customer_city = 'Dublin';
```

00% 30:8

Result Grid Filter Rows: Search Export:

customer_id	customer_name	customer_email	customer_phone_...	customer_city	order_date	order_status	order_id
1	Sarah Murphy	sarah.murphy@gmail.com	01-555-9876	Dublin	2023-04-20 20:00:00	shipped	11
5	Ciara Kelly	ciara.kelly@gmail.com	087-555-1111	Dublin	2023-05-03 14:00:00	shipped	5

Figure 17. tracking orders

In the above query, an internal user may try to track which orders are still in the shipped stage for customers in a particular city such as Dublin. This would also help the logistics/operations team in planning since it provides the initial order date for reference as well. It would be useful in later analysis which can include what the average delay in orders was across different customer cities.

3. Analysing order statistics across different customer cities

```
11  select customer_city, sum(tc.number_of_orders) as total_orders, avg(tc.total_amount_spent) as average_amount from top_customers tc
12  inner join customers c
13  on tc.customer_id = c.customer_id
14  group by 1
15  order by total_orders desc, average_amount desc;
```

00% 49:15

Result Grid Filter Rows: Search Export:

customer_ci...	total_orders	average_amount
Dublin	6	4733.3333
Cork	4	9100.0000
Limerick	4	8500.0000
Galway	3	2150.0000
Belfast	1	1000.0000

Figure 18. analysis of orders across cities

The above query demonstrates how an internal user can summarize order statisitics such as total number of orders and average amount spent across different customer cities. It shows as an example, that although Dublin has had the most number of

orders placed in it, Cork has contributed almost twice the revenue generated through orders on Excelsior as compared to Dublin.

Apart from customer cities, the order statistics can also be studied across different demographics and metrics such as publisher, writer and comic character by joining the top_customers view with relevant tables.

4. Analysing order distribution based on comic book conditions

```
17  select condition_description, count(distinct order_id) as number_of_orders from consolidated_comic_books ccb
18  inner join orders o
19  on ccb.product_id = o.product_id
20  group by 1
21  order by number_of_orders desc;
```

100% 32:21

Result Grid Filter Rows: Search Export:

condition_descript...	number_of_orders
Very Fine(VF)	6
Mint(MT)	5
Near-Mint(NM)	4
Fine(FN)	3

Figure 19. Order distribution analysis based on comic book conditions

The abover query demonstrates how a user can use comic book conditions as a metric to analyse the number of orders placed through Excelsior.

As per the results, it is shown that overall, the most number of orders were placed for comic books which were in Very Fine (VF) condition, followed by Mint and Near-Mint. It hints at the fact that customers are okay with ordering a comic book in a VF condition because they tend to be less expensive as well. Whereas, mint condition comic books could have been bought by comic book enthusiasts who were willing to pay a bit extra.

This analysis can be expanded to genres, publishers and writers as well.

Conclusions

Overall, this document covers the basic database design of Excelsior which included the different kinds of tables, views and procedular elements of the database, along with the different use cases that may be covered.

As Excelsior gathers more data about the comic books, writers and comic characters, the database is expected to grow at a similar pace and with the table structures, constraints and views defined in this document, the database is expected to perform well in terms of performance and relevance of query results.

Going forward, some of the many improvements and features that may be incorporated into the Excelsior database to make it more useful to other applications within the Excelsior ecosystem are as follows:

1. The inventory table may be expanded by adding multiple columns such as purchase price(price at which the comic book was bought by Excelsior) – this would enable a profit/loss analysis to be presented to the board of Excelsior for decision making.

As Excelsior would grow, so would the number of physical warehouses that would keep stock of the comic books. Therefore, within the inventory table, a warehouse_location column may also be added to keep track of stock across different warehouse locations and as well as to do analysis around the warehouse location and the customer location parallely.

2. Apart from English comic books, Excelsior may also decide in the future to consider selling comic books in multiple languages as well, to appeal to a larger customer segment. The language field may be added to the comic book details table and be used for analysis across different existing demographics.

3. Excelsior could build a search query tracker, which would collect information about how frequently and by how many users, a particular comic book, writer, publisher or comic character, was searched through the front-end applications.

By storing this data within seperate tables, the organization would be able to analyze this data and observe whether there are periodic or popularity trends amongst the users for the different kinds of entities listed above.

References

[1] Swathi, Peddyreddy, A Study on SQL - RDBMS Concepts and Database Normalization (August 5, 2020). JASC: Journal of Applied Science and Computations, Volume VII, Issue VIII, August 2020 ISSN NO: 1076-5131, Page no:127-131, Available at SSRN: https://ssrn.com/abstract=4282707

[2] Kromann, F.M. (2018). MySQL Triggers. In: Beginning PHP and MySQL. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-6044-8_30

[3] https://www.milehighcomics.com