HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
**SOICT**
----- ▯ ▯ ▯ -----

# Project Report of Database Lab

## ***Project Name:*** Comic database management system

Lecturer: **TS. Nguyễn Thị Oanh**

Students:

| STT | Name | Student ID |
|---|---|---|
| 1 | Hồ Trần Anh Vũ | 20194885 |
| 2 | Đinh Thế Kiệt | 20194783 |
| 3 | Đồng Văn Toàn | 20194858 |

**Hanoi, 2021**

## Table of Contents

## I. Business Analysis:

We are oriented to create a comics web. The database of this system has two main stages: crawl and store comics into the database and provide some basic interactions between users with our database server.

### 1. Crawling and storing comics into the database:

We intend to crawl data from an open source website and store it into our database. And we have used 4 tables to implement this stage as well as for storing purposes.

- First, we use table authors to store all authors of comics. This table will have 4 attributes correspondingly:
    - <u>author_id</u>
    - name
    - nationality
    - dob

  to determine all the information related to the author of each comic

- We also have a table comic to store all the comics. This table has some important attributes as follow:
    - <u>comic_id</u>
    - name
    - <u>author_id</u>
    - status
    - first_uploaded
    - last_uploaded
    - current_chapter
    - total_view
    - rating

  to determine all the information of every comic.

- We have table chapters to store all chapters of each comic. This table has the following attributes:

    - chapter_id
    - comic_id
    - num_pages
    - update_date
    - chapter_num
    - views
    - likes
    - comments
    - link

  to determine all the information of every chapter of each comic.

- We have table tags to store the genres of each comic related to. This table will have the following attributes:
    - tag_id
    - name
    - description

  to determine all the information of tags, genres of every comic.

- And we have table tagging to represent the relation between comics and tags (n - n relation). This table will have the following attributes:
    - tagging_id
    - tag_id
    - comic_id

One author can provide one or many kinds of comics. The relation here is 1 - n. In table comics, it has a foreign key from comics.author_id reference to authors.author_id. Each comic will also have an attribute “status” to determine which comic is still updated or not. When the comic is finished, its information is still stored in our database.

One comic can have one or many chapters. The relation here is 1 - n. In table chapters, there is a foreign key from chapters.comic_id reference to comics.comic_id. Each chapter also has an attribute update_date to represent the date updated by the author.

One comic can have one or many tags, one tag can also have one or many kinds of comics. Thus, the relation here is n - n, then we use another table tagging in order to represent this relation.

2. **Provide some basic interactions for users**

We provide some basic interactions for users to access the comics web. In our system, users need to create an account to access all functionalities of this comic web. After creating their accounts, users can subscribe to their favourite comics, read comics, give some likes, comments for chapters and follow their favourite authors as well.

Some tables to store all the information related in this stage is:

Table accounts:

- account_id
- display_name
- username
- password
- account_type
- created_time
- email

Table subscribe:

- subscribe_id
- account_id
- comic_id

Table read:

- read_id
- chapter_id
- date
- account_id

Table likes:

- like_id
- account_id

- chapter_id
- date

Table comments:

- comment_id
- date
- chapter_id
- account_id

Table follow:

- follow_id
- author_id
- account_id
- last_read_day

One user can subscribe to many types of comics as well as one comic can be subscribed by a lot of users. Hence, the relation here is n - n. Then we design a table “subscribe” to store all the information related to this relation between two entities.

The other interactions between users and chapters such as: read, like, comment, we have designed 3 corresponding tables to represent this relationship. Because of n - n relation between users and chapters.

One user can follow their favourite authors as well as one author can be followed by a lot of fans (users). Hence, we designed a table “follow” to represent this relationship between users and authors.

## II. Application:

A database serves as back-end database for a comic-reading book

Users can interact directly with the database (find comics, post, rating, comments, etc…) through CSS UI.

Administrators and Moderators are also able to control and keep track of data stored on websites (views, ranking,…)

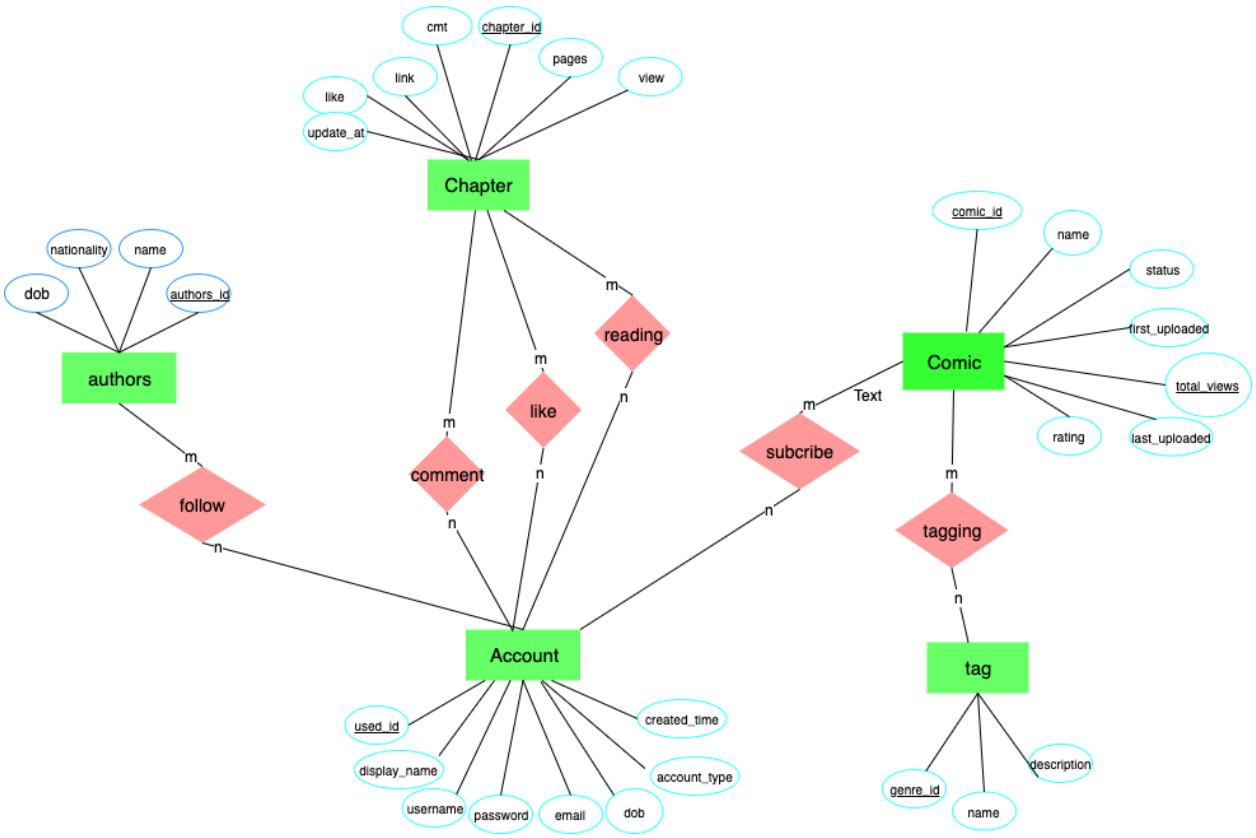
**Stored Data:**

- Collections of comics, include their basic information (authors, chapter, views, publishers…)
- A set of tags (genre) corresponding to each comic.
- List of chapters and link to each chapter of each comic
- Users log in and interact through their personal account.
- The actions can be executed include read, like, comment.

**Featuring:**

- Ranking (top viewed and rated comic)
- Advanced searching (by name of comic, authors, genre tags)
- Filter and sorting (by views, rate, comments, published date) in chosen order

## III. ER Diagram



**IV. Tables details:**

1. Table accounts: Account table to store user account's information.

{

account_id int [pk], primary key to determine an account

display_name varchar(20), the name displayed to others account

username varchar(20), account's username

password varchar(20), account's password

account_type varchar(20), example: vip account, admin account

created_time date, the day account created

email varchar(20), email to contact in case password is lost

}

2. Table authors: Author's table to store author's information

{

author_id int pk, primary key to determine author

name varchar(20), author name published to readers

nationality varchar(20)

dob varchar(20), date of birth of the authors

}

3. Table chapter: Chapter's table to store chapter's information of comics

{

chapter_id int pk, id of each chapter in a comic

comic_id int fk, foreign key to link chapter_id in . id of a comic which the chapter belongs

num_pages int, total pages of the chapter

views int, total views of the chapter

likes int, total likes of the chapter

comments int, total comments of the chapter

link varchar(20), url link to direct to the chapter's content

}

4. Table comic: Comic's table to store comic's information

{

comic_id int pk, primary key to determine comic

name varchar(30), the name of a comic

author_id varchar(20), the author_id of the author's comic

status varchar(20), status of the comic for example: drop, finished, continued

first_uploaded varchar(20), the day that the comic is first uploaded

last_uploaded varchar(20) , the day that the comic is last uploaded

current_chapter varchar(20), the number of chapters is published

total_view int, the sum of each chapter's views

rating varchar(20), a measurement of how good the comic is rated by readers

}

5. Table tags: Tag's table to store comic's information about its genres

{

tag_id varchar(20) pk, id of the tag

name varchar(20), the name of the tag for example: genre, author

description varchar(50), it can describe genre, author

}

**Relation:**

1. Table reads: to store the history of accounts's reading

{

read_id varchar(20) pk, id of the read

chapter_id varchar(20), chapter_id of a chapter that have been read

date varchar(20), the day that the account read the chapter

account_id varchar(20), the account that read the chapter

}

2. Table likes: to store the history of accounts's liking

{

like_id varchar(20) pk, id of the like

chapter_id varchar(20), chapter_id that have been liked

account_id varchar(20) , the account _id that like the chapter

date varchar(20), the day of liking

}

3. Table comments: to store all accounts's comments

{

comment_id varchar(20) pk, id of each comment

date varchar(20), the day user comment

chapter_id varchar(20), identify that chapter that have been commented

account_id varchar(20), the id of the account commenting on the chapter

content varchar(100), the content of comments

}

4. Table tagging: to store the comics's tagging

{

tagging_id varchar(20)  pk, id to identify a tagging

tag_id  varchar(20), id of a tag

comic_id  varchar(20), the id of an comic

}

5. Table following: to store the comics's followings

{

follow_id int pk, id to identify each following

author_id int, id of an author that have been followed

account_id int, id of an account that follow the author

last read day: the last day that the author is read by the account

}

6. Table subscribe: to store the comics's subscribes

{

subscribe_id int pk: id to identify each subscription

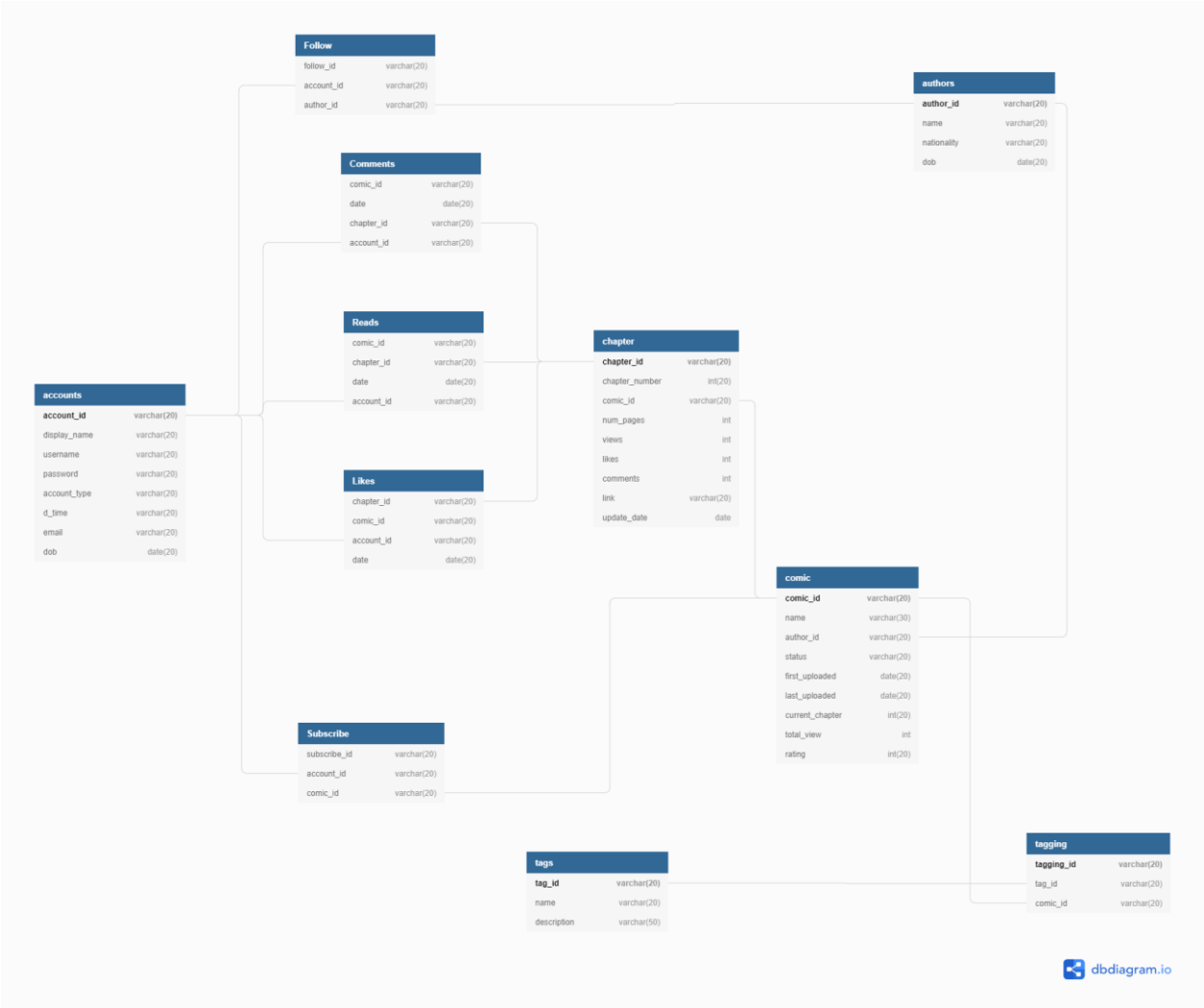
account_id int: id of an account subscribe a comic

comic_id int: id of a comic that have been subscribed

}

## V. Diagram

Link to diagram platform: https://dbdiagram.io/d/60a68c16b29a09603d15cf22




## VI. Query Question

1. Display the comic has  the maximum total views?

comic/postgres@DatabaseLab01

Query Editor  Query History

```
select * from comic
where total_view = (select max(total_view) from comic)
```

Result:

Data Output  Explain  Messages  Notifications

| | comic_id [PK] integer | name character varying (50) | author_id integer | status character varying (50) | first_uploaded date | last_uploaded date | current_chapter integer | total_view integer | rating integer |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | Bleach | 27 | completed | 2012-02-02 | 2015-07-27 | 129 | 979802 | 1 |

2. Display the accounts who follow 'Arthur Conan Doyle'?

```
select a.account_id, a.display_name, a.username from accounts a
join follows f
on f.account_id = a.account_id
join authors au
on f.author_id = au.author_id
where au.name = 'Arthur Conan Doyle'
```

Result:

| | account_id [PK] integer | display_name character varying (50) | username character varying (50) |
|---|---|---|---|
| 1 | 92 | Valaria Garretson | vgarretson2j |
| 2 | 131 | Martyn Fourcade | mfourcade3m |
| 3 | 30 | Craggie Colten | ccoltent |
| 4 | 144 | Joice Snoxell | jsnoxell3z |
| | | | |

3. Display list of all chapters(chapter_id) of all comics?

```
select comic.name, array_agg(distinct chapter_id)
from chapter
join comic
on comic.comic_id = chapter.comic_id
group by comic.name
```

Result:

Data Output | Explain | Messages | Notifications

| | name character varying (50) | array_agg integer[] |
|---|---|---|
| 1 | 20th Century Boys | {9,26,38,44} |
| 2 | Absolute Boyfriend | {21,37,54,58} |
| 3 | Akira | {61} |
| 4 | Azumanga Daioh | {16} |
| 5 | Berserk | {7,24} |
| 6 | Bleach | {4,8,31,35} |
| 7 | Cardcaptor Sakura | {45} |
| 8 | Claymore | {36,42,63,68} |
| 9 | Chobits | {2,6,17,70} |
| 10 | Death Note | {3,33,48} |
| 11 | Dragon Ball | {25,66} |
| 12 | Fruits Basket | {27,29,50} |

4. Display all chapters which were read by 'Parke Aulton'?

```
select a.display_name, array_agg(chapter_id)
from read r
join accounts a
on a.account_id = r.account_id
group by a.display_name having a.display_name = 'Parke Aulton'

```

Result:

| | display_name character varying (50) | array_agg integer[] |
|---|---|---|
| 1 | Parke Aulton | {44,30,38,8,6} |

5. Display the comic with comments on the date '2021-01-06' which have status 'completed'?

```
select com.comic_id, com.name from comments co
join chapter ch
on co.chapter_id = ch.chapter_id
join comic com
on ch.comic_id = com.comic_id
where com.status = 'completed'
and co.date = '2021-01-06'
```

Result:

Data Output | Explain | Messages | Notifications

| comic_id [PK] integer | name character varying (50) |
|---|---|
| | |

6. Display the information of the author whose comic is rated highest?

```
select au.author_id, au.name, co.rating from comic co
join authors au
on co.author_id = au.author_id
where co.rating = (select max(rating) from comic)
```

Result:

Data Output | Explain | Messages | Notifications

| | author_id integer | name character varying (50) | rating integer |
|---|---|---|---|
| 1 | 14 | Catullus | 5 |
| 2 | 29 | Robert Frost | 5 |
| 3 | 20 | Percy Bysshe Shelley | 5 |

7. Display the accounts created between ‘2021-05-02’ and ‘2021-06-02’?

```
select * from accounts
where created_time
between '2021-05-02' and '2021-06-02'

```

Result:

```
select extract(year from NOW())- extract(year from dob) into cnt
```

Data Output | Explain | Messages | Notifications

| account_id [PK] integer | display_name character varying (50) | username character varying (50) | password character varying (50) | account_type character varying (50) | created_time date | email character varying (50) | dob character varying (50) |
|---|---|---|---|---|---|---|---|

8. Display all the comics with the tag name 'Adventure’'?

```
select tags.name, array_agg(co.name) from tagging t
join comic co
on t.comic_id = co.comic_id
join tags
on t.tag_id = tags.tag_id
group by tags.name having tags.name = 'Adventure'

```

Result:

Data Output Explain Messages Notifications

| | name character varying (50) | array_agg character varying[] |
|---|---|---|
| 1 | Adventure | {"Fullmetal Alchemist ","Vampire Knight",Gantz,"Cardcaptor Sakura",Monster,"Ranma ",Claymore} |

9. Assume that age of users must be >= 16, write a trigger that guarantees this constraint?

```
create trigger tg_check_insert
before insert on accounts
for each row
execute procedure tf_check_insert();

create or replace function tf_check_insert() returns trigger as
$$
declare
    cnt int;
begin
    select extract(year from NOW())-extract(year from dob) into cnt
    from accounts
    where account_id = NEW.account_id;

    if (cnt < 16) then
    return NULL;
    else
    return NEW;
    end if;
end;
$$
language plpgsql;
```

10. Assume that the number of users must be less than or equal 500, write a trigger that guarantees this constraint?

```
create trigger tg_check_num_insert
before insert on accounts
for each row
execute procedure tf_check_num_insert();

create or replace function tf_check_num_insert() returns trigger as
$$
declare
    cnt int;
begin
    select count(*) into cnt
    from accounts
    where account_id = NEW.account_id;

    if (cnt <= 500) then
        return NEW;
    else
        return NULL;
    end if;
end;
$$
language plpgsql;
```

11. Write a SQL query to fetch “NAME” from the Comic table using the alias name as <comic name>.

select name as Comic_Name from comic

Result:

Data Output Explain Messages N

| | comic_name character varying (50) |
|---|---|
| 1 | Death Note |
| 2 | Naruto |
| 3 | Bleach |
| 4 | Fullmetal Alchemist |
| 5 | Fruits Basket |
| 6 | Berserk |
| 7 | Love Hina |
| 8 | One Piece |
| 9 | Rurouni Kenshin |
| 10 | Chobits |
| 11 | Azumanga Daioh |
| 12 | Monster |
| 13 | Tsubasa, RESERVoir CHRoNi... |
| 14 | Yotsuba&! |
| 15 | Ranma |

12. Create view displaying chapter account named "Jillana Ealam" have liked

create view abc as

select accounts.display_name, chapter.chapter_id from likes

join chapter on likes.chapter_id = chapter.chapter_id

join accounts on accounts.account_id = likes.account_id

where accounts.display_name = 'Jillana Ealam'

Result:

Query Editor  Query History

```
create view abc as
    select accounts.display_name, chapter.chapter_id from likes
    join chapter on likes.chapter_id = chapter.chapter_id
    join accounts on accounts.account_id = likes.account_id
    where accounts.display_name = 'Jillana Ealam'
```

Data Output  Explain  Messages  Notifications

```
CREATE VIEW

Query returned successfully in 44 msec.
```

13. Display the comic by tag

    select comic.name, tags.name

    from tagging join tags on tagging.tag_id = tags.tag_id

    join comic on comic.comic_id = tagging.comic_id

    Result:

Data Output Explain Messages Notifications

| | name character varying (50) | name character varying (50) |
|---|---|---|
| 1 | Chobits | Action |
| 2 | Cardcaptor Sakura | Cyberpunk and derivatives |
| 3 | Fullmetal Alchemist | Adventure |
| 4 | xxxHOLiC | Speculative |
| 5 | Vampire Knight | Adventure |
| 6 | GTO | Comedy |
| 7 | Bleach | Action |
| 8 | Ouran High School Host Club | Speculative |
| 9 | Cardcaptor Sakura | Satire |
| 10 | Dragon Ball | Science fiction |
| 11 | Berserk | Horror |
| 12 | Nausicaä of the Valley of the... | Crime and mystery |
| 13 | Inuyasha | Crime and mystery |
| 14 | Naruto | Comedy |
| 15 | Bleach | Science fiction |

14. Display a comic that is followed by the largest number of accounts.

```
select comic.name, count(subscribe.comic_id)
from subscribe join comic on subscribe.comic_id = comic.comic_id
group by  comic.comic_id
```

Result:

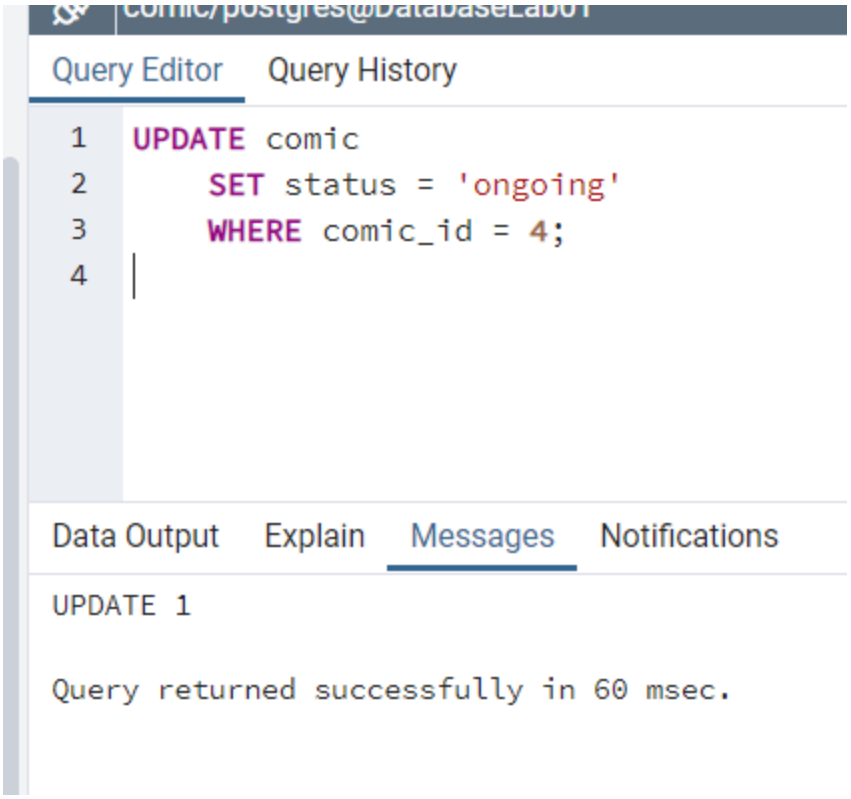
Data Output | Explain | Messages | Notificatic

| | name character varying (50) | count bigint |
|---|---|---|
| 1 | Vampire Knight | 2 |
| 2 | Fullmetal Alchemist | 3 |
| 3 | Chobits | 3 |
| 4 | Berserk | 2 |
| 5 | Yotsuba&! | 7 |
| 6 | Claymore | 1 |
| 7 | Tsubasa, RESERVoir CHRoNi... | 2 |
| 8 | Naruto | 3 |
| 9 | Dragon Ball | 2 |
| 10 | Azumanga Daioh | 4 |
| 11 | Love Hina | 5 |
| 12 | Rurouni Kenshin | 6 |
| 13 | Ranma | 4 |
| 14 | Monster | 2 |
| 15 | Cardcaptor Sakura | 6 |
| 16 | Ouran High School Host Club | 1 |

15. Update a status of comic_id= 4 to ongoing

```
UPDATE comic
SET status = 'ongoing'
WHERE comic_id = 4;
```

Result:




16. A function to update status from "delayed" to "ongoing" (when an chapter is updated)

```
create or replace function public.abcd(in a int) returns void as
$$
begin
        UPDATE comic
        SET status = 'ongoing'
        WHERE comic_id = a;
end;
$$
language plpgsql

select public.abcd(7)
drop function public.abcd
```

select * from comic

Query Editor  Query History

```
create or replace function public.abcd(in a int) returns void as
$$
begin
    UPDATE comic
    SET status = 'ongoing'
    WHERE comic_id = a;
end;
$$
language plpgsql

select public.abcd(7)
drop function public.abcd

select * from comic
```

Data Output  Explain  Messages  Notifications

| | abcd void |
|---|---|
| 1 | |

17. Display a comic that have been last updated between '2004-01-01' and '2004-01-31'

select * from comic

where last_uploaded > '2013-01-01' and last_uploaded <'2016-01-31'

Result:

Data Output Explain Messages Notifications

| | comic_id [PK] integer | name character varying (50) | author_id integer | status character varying (50) | first_uploaded date | last_uploaded date | current_chapter integer | total_view integer | rating integer |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | Bleach | 27 | completed | 2012-02-02 | 2015-07-27 | 129 | 979802 | 1 |
| 2 | 14 | Yotsuba&! | 29 | completed | 2013-06-11 | 2015-11-24 | 149 | 747796 | 1 |
| 3 | 23 | Gantz | 29 | completed | 2010-08-22 | 2015-11-06 | 5 | 321801 | 3 |
| 4 | 7 | Love Hina | 16 | ongoing | 2014-05-17 | 2015-06-03 | 14 | 233367 | 2 |

18. Delete comments that contains sensitive word 'cat'

delete from comments

where content like '%cat%'

Result:

Query Editor Query History

```
delete from comments
    where content like '%cat%'
```

Data Output Explain Messages Notifications

```
DELETE 1

Query returned successfully in 47 msec.
```

19. Display the 5 most recent comic accounts "Jillana Ealam" have read.

```
select * from read
join accounts on read.account_id = accounts.account_id
where accounts.display_name = 'Jillana Ealam'
order by date asc
LIMIT 5
```

Result:

Data Output | Explain | Messages | Notifications

| | read_id integer | chapter_id integer | date date | account_id integer | account_id integer | display_name character varying (50) | username character varying (50) | password character varying (50) | account_type character varying (50) | created_time date | emai char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 408 | 41 | 2020-07... | 4 | 4 | Jillana Ealam | jealam3 | oP74CmLqVJUr | normal | 2017-04-10 | jeala |
| 2 | 383 | 50 | 2020-07... | 4 | 4 | Jillana Ealam | jealam3 | oP74CmLqVJUr | normal | 2017-04-10 | jeala |
| 3 | 432 | 3 | 2021-06... | 4 | 4 | Jillana Ealam | jealam3 | oP74CmLqVJUr | normal | 2017-04-10 | jeala |

20. A trigger to delete comments that contain sensitive word such as "cat"

```
create trigger afdelete0
after insert on comments
for each row
execute procedure tf_af_delete();

create or replace function tf_af_delete() returns trigger as $$
begin
        delete from comments
        where comments.content like '%cat%';
        return old;
end;
$$ language plpgsql
```

drop trigger afdelete0 on comments

insert into comments (comment_id, chapter_id, account_id, date, content)

values (1001, 36, 91, '2021-04-01', 'asfcatasf');

Result:

```
insert into comments (comment_id, chapter_id, account_id, date, content)
values (1001, 36, 91, '2021-04-01', 'asfcatasf');
```

Data Output | Explain | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 41 msec.
```

21. Detecting comments with content includes restricted words (zero).

SELECT comment_id, account_id, username, content

FROM comments AS c NATURAL JOIN accounts AS a

WHERE c.content LIKE '%zero%'

Result:

Data Output | Explain | Messages | Notifications

| | comment_id integer | account_id integer | username character varying (50) | content character varying (100) |
|---|---|---|---|---|
| 1 | 34 | 138 | mfielden3t | zero tolerance |

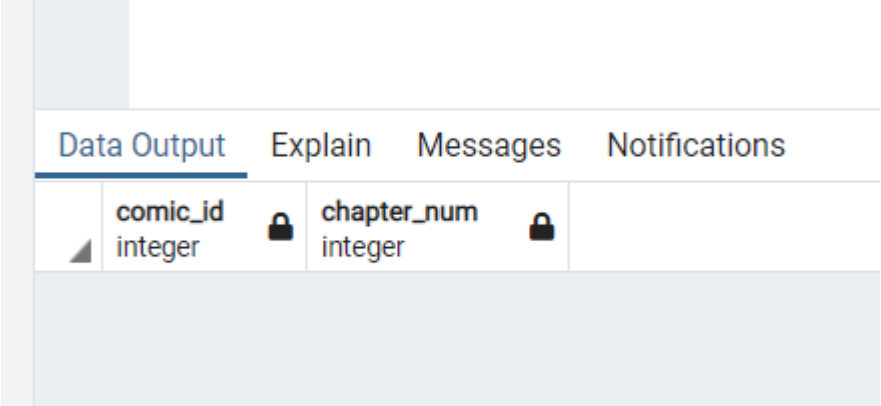
22. Look for every unread chapter of all comics followed by an account(account_id=3).

```
SELECT  ch.comic_id, ch.chap_num
FROM accounts a
        JOIN subscribe s on a.account_id = s.account_id
        JOIN chapter ch on s.comic_id = ch.comic_id
WHERE a.account_id = 3

EXCEPT

SELECT  c.comic_id, ch.chap_num
FROM comic c
        JOIN chapter ch on c.comic_id = ch.comic_id
        JOIN read r on ch.chapter_id = r.chapter_id
        JOIN accounts a on r.account_id = a.account_id
WHERE a.account_id = 3
```

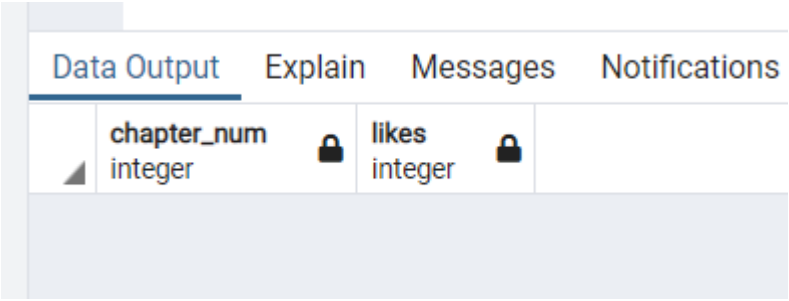
Result:




23. Find the chapter with the highest likes in a comic(comic_id=1).

```
SELECT chap_num,likes
FROM chapter
WHERE likes =
(
        SELECT MAX(likes)
        FROM chapter
        LIMIT 5
)
```

Result:




24. [Trigger] When an account reads the latest chapter of any comic.

```
CREATE TRIGGER like_up
AFTER INSERT
ON likes
FOR EACH ROW
WHEN (NEW.like_id IS NOT NULL)
EXECUTE PROCEDURE like_up();
```

25. Look for the very last reading of an account(account_id = 3).

```
SELECT name, chap_num, date
FROM  read NATURAL JOIN chapter NATURAL JOIN comic
WHERE date = (SELECT MAX(date)
              FROM read
              WHERE account_id=3)
```

Result:

Query Editor  Query History

```
SELECT name, chapter_num, date
FROM  read NATURAL JOIN chapter NATURAL JOIN comic
WHERE date = (SELECT MAX(date)
              FROM read
              WHERE account_id=3)
```

26. Look for the most replicated (favourite) tag of all comic that an account subscribe to (account_id =3).

```
SELECT t.name, COUNT(name)
FROM accounts a
    JOIN subscribe s on a.account_id = s.account_id
    JOIN tagging tg on s.comic_id = tg.comic_id
    JOIN tags t on tg.tag_id= t.tag_id
```

```
WHERE a.account_id =3
GROUP BY name
LIMIT 1
```

Result:

Data Output | Explain | Messages | Notifications

| | name character varying (50) | count bigint |
|---|---|---|
| 1 | Adventure | 1 |

27. Find if an account had subscribed to more than one comic belonging to the same author(account_id=3).

```
SELECT a.name,count(a.name)
FROM subscribe s
        JOIN comic c on s.comic_id = c.comic_id
        JOIN authors a on c.author_id = a.author_id
WHERE account_id=3
GROUP BY a.name
HAVING count(a.name) >1
```

Result:

```
SELECT a.name,count(a.name)
FROM subscribe s
    JOIN comic c on s.comic_id = c.comic_id
    JOIN authors a on c.author_id = a.author_id
WHERE account_id=3
GROUP BY a.name
HAVING count(a.name) >1
```

28. Find all the comics that still have been updated from the year 2019 until now.

SELECT DISTINCT comic.name, update_date

FROM comic NATURAL JOIN chapter

WHERE chapter.update_date > '2019-01-01'

Result:

Query Editor Query History

```
SELECT DISTINCT comic.name, update_date
FROM comic NATURAL JOIN chapter
WHERE chapter.update_date > '2019-01-01'
```

29. Show ranking of most to least popular authors(based on amount of followers) on a given tag(tag.name=Action).

SELECT a.name, count(a.name)

```
FROM authors a
        JOIN comic c on a.author_id = c.author_id
        JOIN tagging tg on c.comic_id = tg.comic_id
        JOIN tags t on tg.tag_id =t.tag_id
WHERE t.name='Action'
GROUP BY a.name
ORDER BY count(a.name) DESC
```

Result:

Data Output | Explain | Messages | Notificati

| | name character varying (50) | count bigint |
|---|---|---|
| 1 | George Gordon Byron | 2 |
| 2 | Robert Frost | 2 |

30. Show the ranking of all the unfinished comics of a given author(author.name=Seneca).

```
SELECT c.name, c.rating, c.status
FROM authors a
        JOIN comic c on a.author_id = c.author_id
WHERE a.name='Catullus' AND  NOT c.status  =  'completed'
ORDER BY rating DESC
```

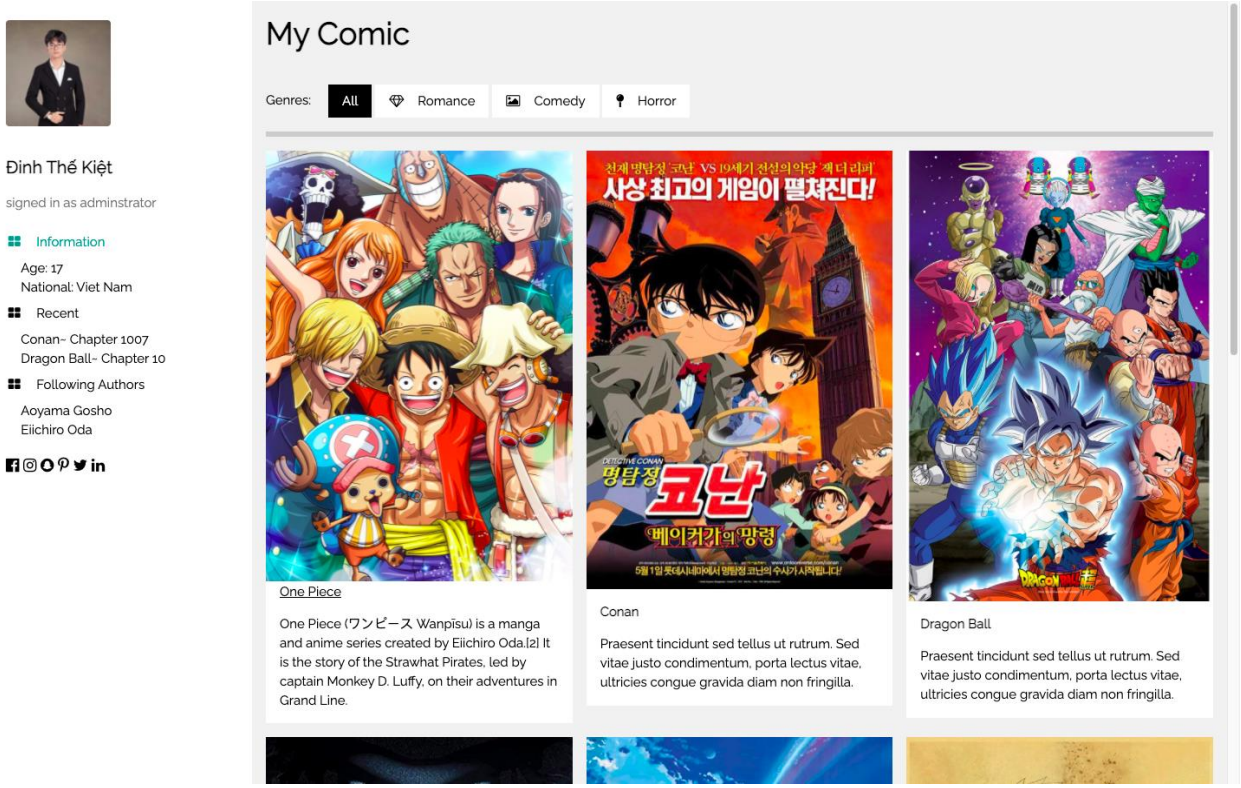
Results:

Data Output | Explain | Messages | Notifications

| | name character varying (50) | rating integer | status character varying (50) |
|---|---|---|---|
| 1 | Rurouni Kenshin | 5 | ongoing |
| 2 | Fruits Basket | 3 | ongoing |

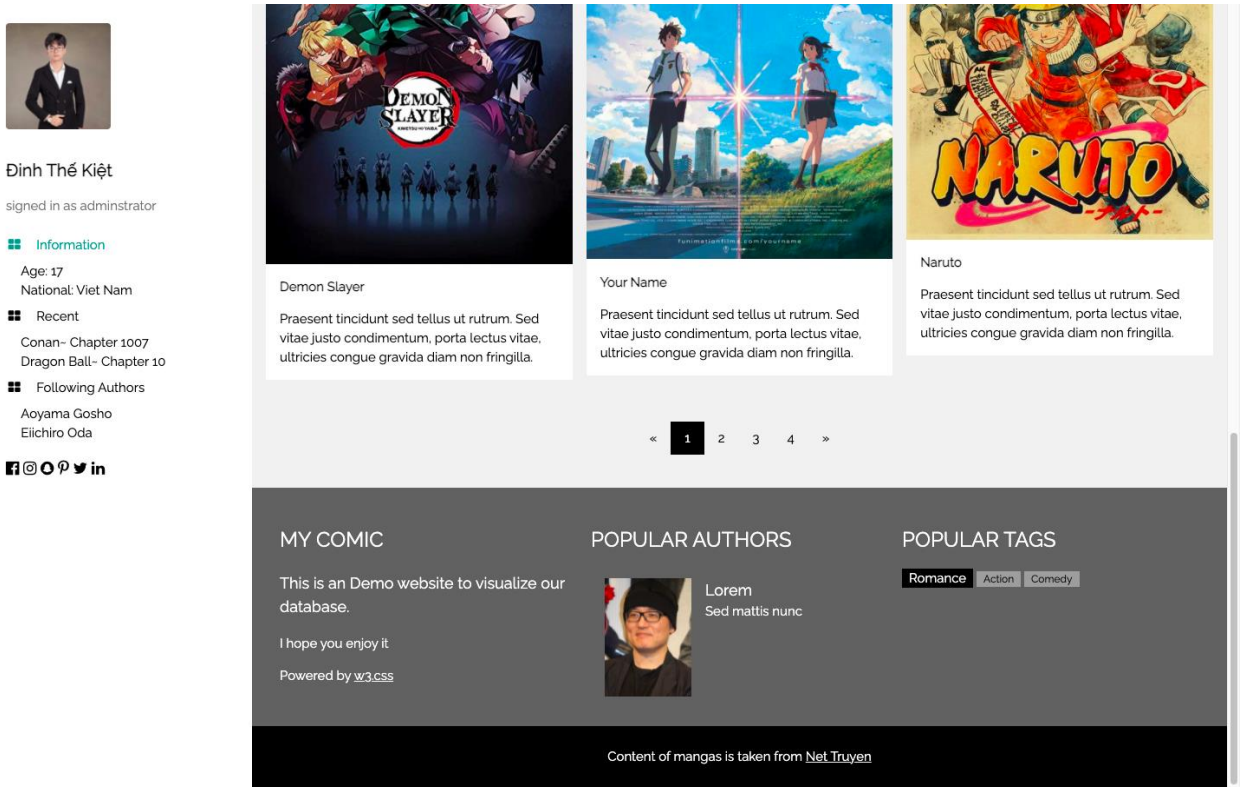
## VII. Graphical user interface.

We have prepared a demo website to visualize our database.



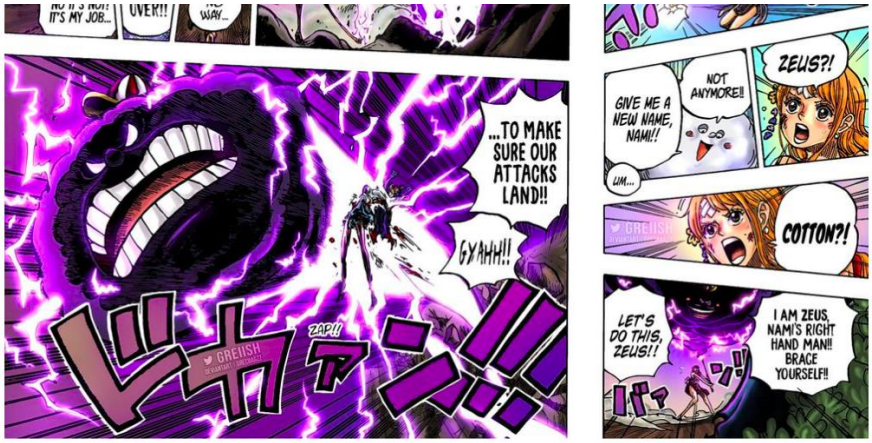

On the left side of the website, it contains the information of a user such as personal information, recent chapter user have read and authors that the user follow.

On the right side, it shows some comics which are grouped by genre(tag) to let the user find the comic easily.




At the end of our main website, a brief introduction of my website, popular authors, and popular tags are shown.

You can also click into a chapter to see the content of the comic's chapter.







This is the end of chapter 1096

Like: 2006 likes

Rated 3.5/5

Comment

- Account number 1:

  dåk dåk

- Account number 2:

  bruh bruh

At the end of each chapter, we have space to let the user interact with each other by liking the chapter or comment their experience in the comment section.