**요구사항 정의서**

\* 목차

1. . 프로젝트 개요

1.1. 프로젝트 소개

1.2. 프로젝트 배경

1.3. 프로젝트 시나리오

1.4. 업무 배경도

2. 프로젝트 구현

2.1 ERD

2.2 릴레이션 스키마

3. 구현

3.1 릴레이션

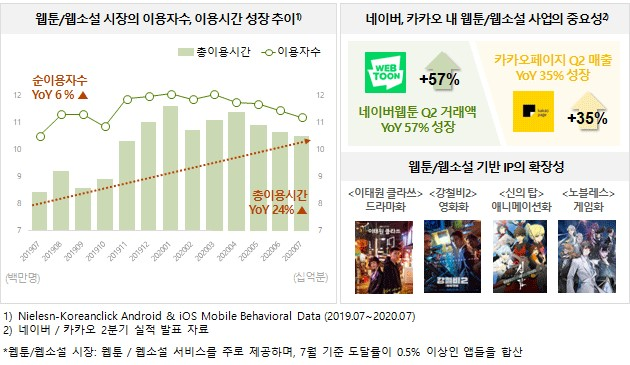
3.2 기능별 쿼리

3.3 시스템 아키텍쳐

**1.1. 프로젝트 소개**

저희 서비스는 사용자가 본 웹툰/웹소설에 대해 별점 및 리뷰를 남기고 별점에 따라 나의 취향에 맞는 작품을 추천해주는 서비스입니다.  
또한 매주 새롭게 올라오는 회차에 대한 라이브톡을 진행하여 독자 모두가 모여 이야기하며 작품에 대한 감상을 나눌 수 있습니다.

**1.2. 프로젝트 배경**



출처 : 닐슨코리아클릭 | 문화체육관광부 보도자료

현재 웹툰 및 웹소설을 드라마로 제작한 작품들이 성공을 거두며, 이로 인해 원작에 대한 관심도가 상승하고 있습니다. 닐슨코리아클릭의 통계에 따르면, 2020년 7월에 웹툰과 웹소설의 총 이용시간은 전년 동월 대비 24% 증가했습니다. 또한 문체부의 보고에 따르면, 웹소설 산업의 규모는 1조 390억 원으로 나타났습니다. 하지만 규모에 비해 작품 정보를 얻을 수 있는 충분한 곳이 부족한 상황입니다.

우리는 동일한 작품을 공유하는 사용자 간에는 서로 다른 시각과 해석을 나누며 다양한 관점을 탐험하여 더욱 만족스러운 작품 감상을 즐길 수 있습니다.

**1.3. 프로젝트 시나리오**

1. 사용자가 회원가입을 한다.

o 이메일를 입력합니다.

o 패스워드를 입력합니다.

o 회원가입을 완료합니다.

2. 사용자가 인증받은 이메일과 패스워드를 입력하여 로그인합니다.

3. 사용자는 마이페이지에 접속합니다.

o 마이페이지에서는 닉네임을 설정합니다.

o 마이페이지에서는 내가 평가한 작품, 나를 팔로우하는 회원 목록, 내가 팔로잉한 회원 목록을 확인 할 수 있습니다.

4. 작품을 선택하여 들어갑니다.

o 작품에 대한 코멘트와 별점을 남길 수 있습니다.

 코멘트 글자 수는 200자를 넘길 수 없습니다.

 욕설/비방어는 사용할 수 없습니다.

o 작품에 대한 회차별 또는 전체 작품에 대한 별점을 남길 수 있습니다.

o 전체 작품의 전체 평균 별점을 확인할 수 있습니다.

o 작품의 평가 상태를 확인할 수 있습니다.

o 작품명, 작가를 검색하여 작품을 찾을 수 있습니다.

5. 사용자는 자신이 보는 웹툰/웹소설이 업로드 되는 날에 라이브톡에 들어갑니다.

o 라이브톡은 상위 n개의 작품들에서만 진행됩니다.

o 라이브톡에는 회원들만 참여가 가능합니다.

o 12시가 되면 이전의 채팅들이 사라집니다.

o 채팅들이 사라지기 전에 사용자들은 채팅 내용을 백업할 수 있습니다.

o 라이브톡에서 욕설/비방어를 입력할 경우 채팅을 보낼 수 없다는 메세지가 뜹니다.

o 라이브톡은 상시로 참여할 수 있습니다.

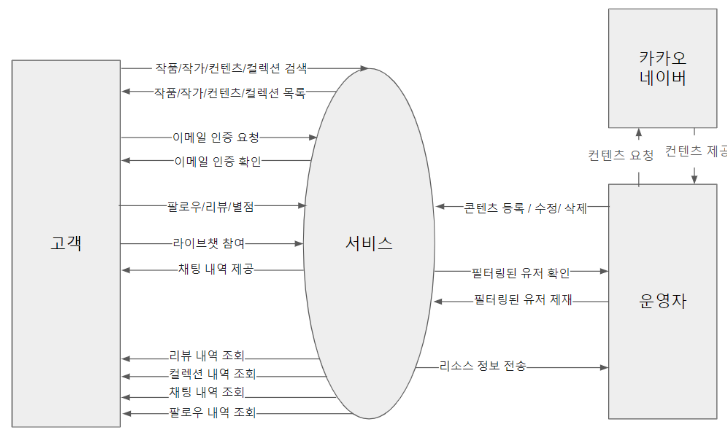
6. 사용자는 자신이 좋아하는 작품을 클릭해 해당 작품이 담겨있는 컬렉션을 봅니다.

o 사용자는 작품들을 담아서 컬렉션들을 만들 수 있습니다.

o 회원 닉네임, 컬렉션 이름을 검색하여 다른 사용자가 만든 컬렉션을 볼 수 있습니다.

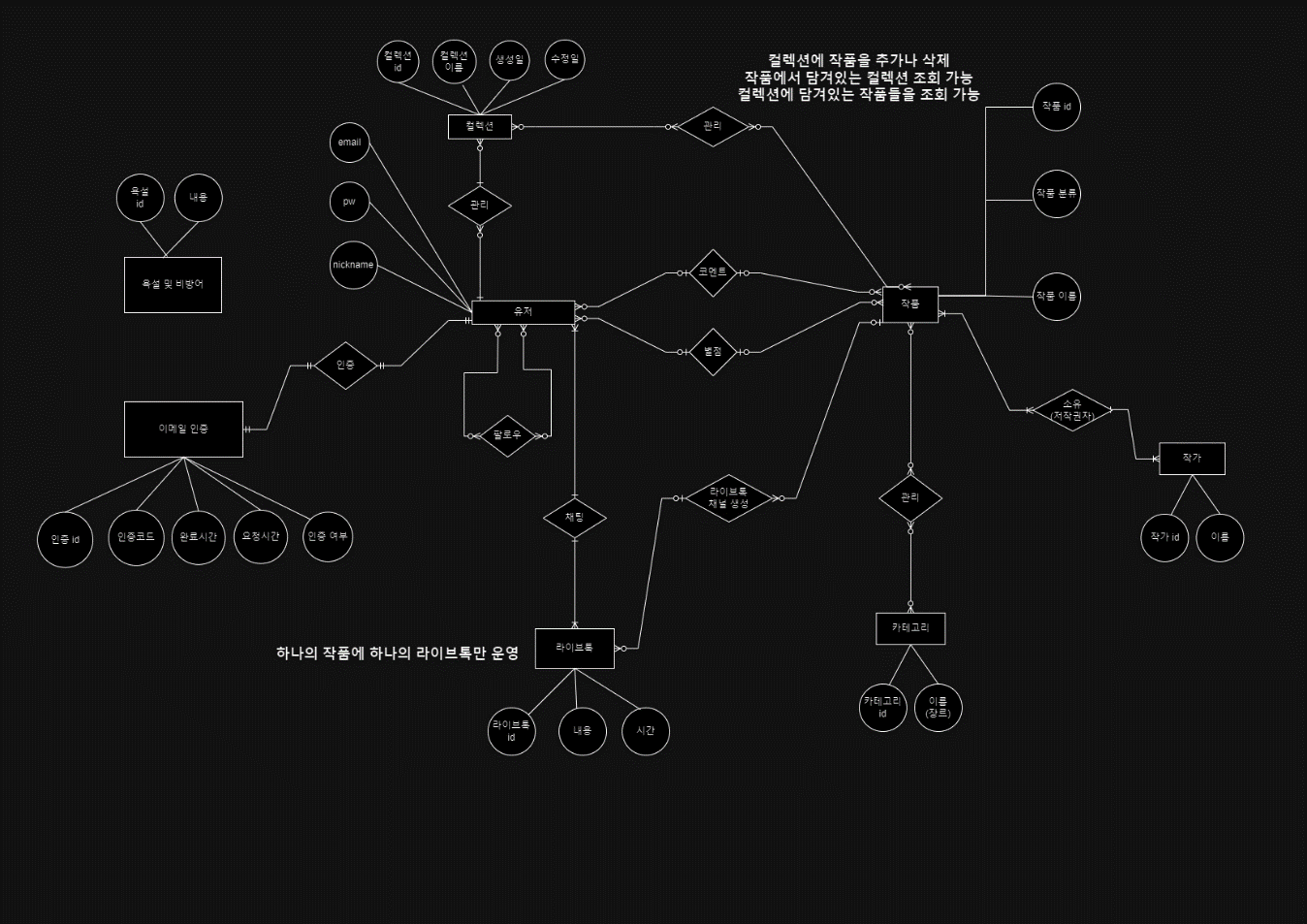
o 작품이 담겨있는 다른 사용자들의 컬렉션을 볼 수 있습니다.

**1.4. 업무 배경도**

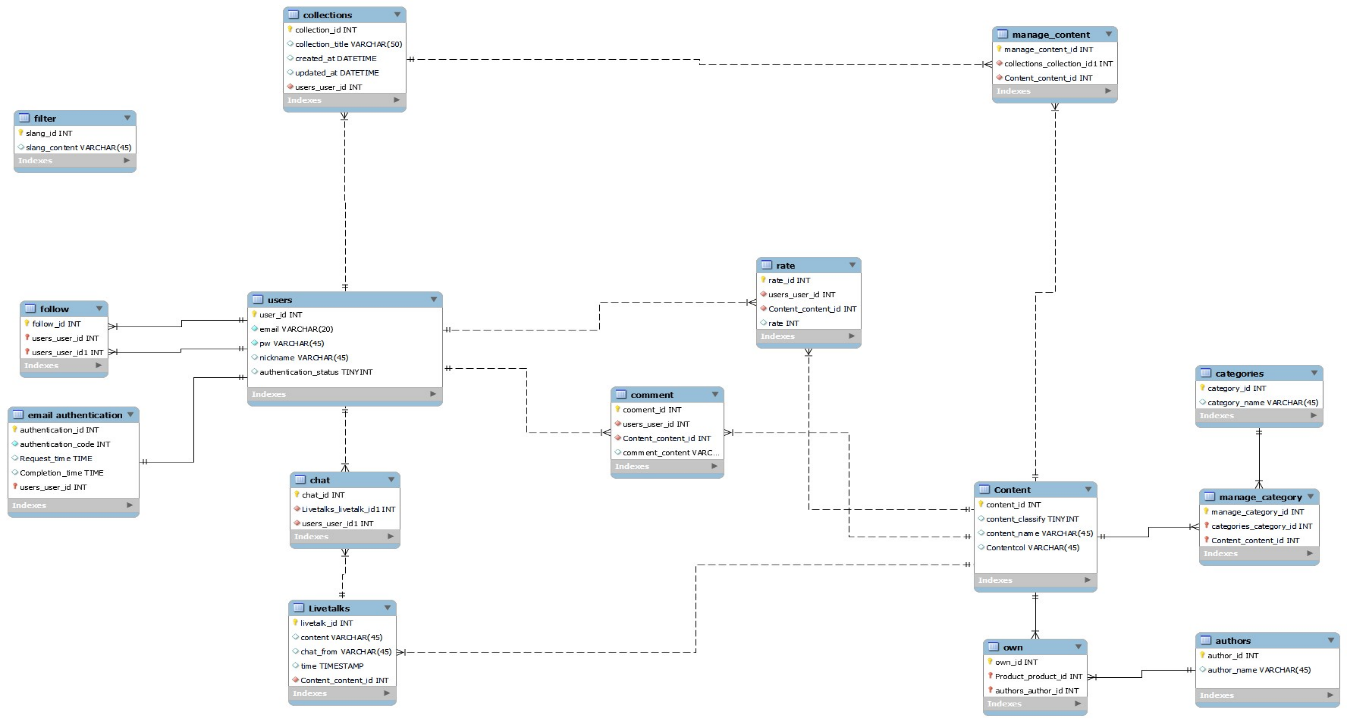


**2. 설계**

**2-1. ERD**



**2.2 릴레이션 스키마**



**3. 구현**

**3.1 릴레이션**

|  |
| --- |
| **user** |
| CREATE TABLE IF NOT EXISTS `mydb`.`users` (  `user\_id` INT NOT NULL AUTO\_INCREMENT,  `email` VARCHAR(20) NOT NULL,  `pw` VARCHAR(45) NOT NULL,  `nickname` VARCHAR(45) NULL,  `authentication\_status` TINYINT NULL,  PRIMARY KEY (`user\_id`))  ENGINE = InnoDB; |
| **collection** |
| CREATE TABLE IF NOT EXISTS `mydb`.`collections` (  `collection\_id` INT NOT NULL AUTO\_INCREMENT,  `collection\_title` VARCHAR(50) NULL,  `created\_at` DATETIME NULL,  `updated\_at` DATETIME NULL,  `users\_user\_id` INT NOT NULL,  PRIMARY KEY (`collection\_id`),  INDEX `fk\_collections\_users1\_idx` (`users\_user\_id` ASC) VISIBLE,  CONSTRAINT `fk\_collections\_users1`  FOREIGN KEY (`users\_user\_id`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

|  |
| --- |
| **content** |
| CREATE TABLE IF NOT EXISTS `mydb`.`Content` (  `content\_id` INT NOT NULL AUTO\_INCREMENT,  `content\_classify` TINYINT NULL,  `content\_name` VARCHAR(45) NULL,  `Contentcol` VARCHAR(45) NULL,  PRIMARY KEY (`content\_id`))  ENGINE = InnoDB; |
| **category** |
| CREATE TABLE IF NOT EXISTS `mydb`.`categories` (  `category\_id` INT NOT NULL AUTO\_INCREMENT,  `category\_name` VARCHAR(45) NULL,  PRIMARY KEY (`category\_id`))  ENGINE = InnoDB; |
| **Author** |
| CREATE TABLE IF NOT EXISTS `mydb`.`authors` (  `author\_id` INT NOT NULL AUTO\_INCREMENT,  `author\_name` VARCHAR(45) NULL,  PRIMARY KEY (`author\_id`))  ENGINE = InnoDB; |

|  |
| --- |
| **livetalk** |
| CREATE TABLE IF NOT EXISTS `mydb`.`Livetalks` (  `livetalk\_id` INT NOT NULL AUTO\_INCREMENT,  `content` VARCHAR(45) NULL,  `chat\_from` VARCHAR(45) NULL,  `time` TIMESTAMP NULL,  `Content\_content\_id` INT NOT NULL,  PRIMARY KEY (`livetalk\_id`),  INDEX `fk\_Livetalks\_Content1\_idx` (`Content\_content\_id` ASC) VISIBLE,  CONSTRAINT `fk\_Livetalks\_Content1`  FOREIGN KEY (`Content\_content\_id`)  REFERENCES `mydb`.`Content` (`content\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |
| **Chat** |
| CREATE TABLE IF NOT EXISTS `mydb`.`chat` (  `chat\_id` INT NOT NULL AUTO\_INCREMENT,  `Livetalks\_livetalk\_id1` INT NOT NULL,  `users\_user\_id1` INT NOT NULL,  INDEX `fk\_Livetalks\_has\_users\_Livetalks2\_idx` (`Livetalks\_livetalk\_id1` ASC) VISIBLE,  INDEX `fk\_Livetalks\_has\_users\_users2\_idx` (`users\_user\_id1` ASC) VISIBLE,  PRIMARY KEY (`chat\_id`),  CONSTRAINT `fk\_Livetalks\_has\_users\_Livetalks2`  FOREIGN KEY (`Livetalks\_livetalk\_id1`)  REFERENCES `mydb`.`Livetalks` (`livetalk\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Livetalks\_has\_users\_users2`  FOREIGN KEY (`users\_user\_id1`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

|  |
| --- |
| **follow** |
| CREATE TABLE IF NOT EXISTS `mydb`.`follow` (  `follow\_id` INT NOT NULL AUTO\_INCREMENT,  `users\_user\_id` INT NOT NULL,  `users\_user\_id1` INT NOT NULL,  PRIMARY KEY (`follow\_id`, `users\_user\_id`, `users\_user\_id1`),  INDEX `fk\_users\_has\_users\_users2\_idx` (`users\_user\_id1` ASC) VISIBLE,  INDEX `fk\_users\_has\_users\_users1\_idx` (`users\_user\_id` ASC) VISIBLE,  CONSTRAINT `fk\_users\_has\_users\_users1`  FOREIGN KEY (`users\_user\_id`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_users\_has\_users\_users2`  FOREIGN KEY (`users\_user\_id1`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |
| **filter** |
| CREATE TABLE IF NOT EXISTS `mydb`.`filter` (  `slang\_id` INT NOT NULL AUTO\_INCREMENT,  `slang\_content` VARCHAR(45) NULL,  PRIMARY KEY (`slang\_id`))  ENGINE = InnoDB; |

|  |
| --- |
| **Manage content** |
| CREATE TABLE IF NOT EXISTS `mydb`.`manage\_content` (  `manage\_content\_id` INT NOT NULL AUTO\_INCREMENT,  `collections\_collection\_id1` INT NOT NULL,  `Content\_content\_id` INT NOT NULL,  INDEX `fk\_collections\_has\_Product\_collections2\_idx` (`collections\_collection\_id1` ASC) VISIBLE,  INDEX `fk\_collections\_has\_Product\_Product2\_idx` (`Content\_content\_id` ASC) VISIBLE,  PRIMARY KEY (`manage\_content\_id`),  CONSTRAINT `fk\_collections\_has\_Product\_collections2`  FOREIGN KEY (`collections\_collection\_id1`)  REFERENCES `mydb`.`collections` (`collection\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_collections\_has\_Product\_Product2`  FOREIGN KEY (`Content\_content\_id`)  REFERENCES `mydb`.`Content` (`content\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |
| **email authentication** |
| CREATE TABLE IF NOT EXISTS `mydb`.`email authentication` (  `authentication\_id` INT NOT NULL AUTO\_INCREMENT,  `authentication\_code` INT NOT NULL,  `Request\_time` TIME NULL,  `Completion\_time` TIME NULL,  `users\_user\_id` INT NOT NULL,  PRIMARY KEY (`authentication\_id`, `users\_user\_id`),  INDEX `fk\_email authentication\_users1\_idx` (`users\_user\_id` ASC) VISIBLE,  CONSTRAINT `fk\_email authentication\_users1`  FOREIGN KEY (`users\_user\_id`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

|  |
| --- |
| **Manage category** |
| CREATE TABLE IF NOT EXISTS `mydb`.`manage\_category` (  `manage\_category\_id` INT NOT NULL AUTO\_INCREMENT,  `categories\_category\_id` INT NOT NULL,  `Content\_content\_id` INT NOT NULL,  PRIMARY KEY (`manage\_category\_id`, `categories\_category\_id`, `Content\_content\_id`),  INDEX `fk\_categories\_has\_Content\_Content1\_idx` (`Content\_content\_id` ASC) VISIBLE,  INDEX `fk\_categories\_has\_Content\_categories1\_idx` (`categories\_category\_id` ASC) VISIBLE,  CONSTRAINT `fk\_categories\_has\_Content\_categories1`  FOREIGN KEY (`categories\_category\_id`)  REFERENCES `mydb`.`categories` (`category\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_categories\_has\_Content\_Content1`  FOREIGN KEY (`Content\_content\_id`)  REFERENCES `mydb`.`Content` (`content\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |
| **Own** |
| CREATE TABLE IF NOT EXISTS `mydb`.`own` (  `own\_id` INT NOT NULL AUTO\_INCREMENT,  `Product\_product\_id` INT NOT NULL,  `authors\_author\_id` INT NOT NULL,  PRIMARY KEY (`own\_id`, `Product\_product\_id`, `authors\_author\_id`),  INDEX `fk\_Product\_has\_authors\_authors1\_idx` (`authors\_author\_id` ASC) VISIBLE,  INDEX `fk\_Product\_has\_authors\_Product1\_idx` (`Product\_product\_id` ASC) VISIBLE,  CONSTRAINT `fk\_Product\_has\_authors\_Product1`  FOREIGN KEY (`Product\_product\_id`)  REFERENCES `mydb`.`Content` (`content\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Product\_has\_authors\_authors1`  FOREIGN KEY (`authors\_author\_id`)  REFERENCES `mydb`.`authors` (`author\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

|  |
| --- |
| **comment** |
| CREATE TABLE IF NOT EXISTS `mydb`.`comment` (  `cooment\_id` INT NOT NULL AUTO\_INCREMENT,  `users\_user\_id` INT NOT NULL,  `Content\_content\_id` INT NOT NULL,  `comment\_content` VARCHAR(200) NULL,  INDEX `fk\_users\_has\_Content\_users1\_idx` (`users\_user\_id` ASC) VISIBLE,  INDEX `fk\_users\_has\_Content\_Content1\_idx` (`Content\_content\_id` ASC) VISIBLE,  PRIMARY KEY (`cooment\_id`),  CONSTRAINT `fk\_users\_has\_Content\_users1`  FOREIGN KEY (`users\_user\_id`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_users\_has\_Content\_Content1`  FOREIGN KEY (`Content\_content\_id`)  REFERENCES `mydb`.`Content` (`content\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

|  |
| --- |
| **rate** |
| CREATE TABLE IF NOT EXISTS `mydb`.`rate` (  `rate\_id` INT NOT NULL AUTO\_INCREMENT,  `users\_user\_id` INT NOT NULL,  `Content\_content\_id` INT NOT NULL,  `rate` INT NULL DEFAULT NULL,  INDEX `fk\_users\_has\_Content1\_users1\_idx` (`users\_user\_id` ASC) VISIBLE,  INDEX `fk\_users\_has\_Content1\_Content1\_idx` (`Content\_content\_id` ASC) VISIBLE,  PRIMARY KEY (`rate\_id`),  CONSTRAINT `fk\_users\_has\_Content1\_users1`  FOREIGN KEY (`users\_user\_id`)  REFERENCES `mydb`.`users` (`user\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_users\_has\_Content1\_Content1`  FOREIGN KEY (`Content\_content\_id`)  REFERENCES `mydb`.`Content` (`content\_id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB;  SET SQL\_MODE=@OLD\_SQL\_MODE;  SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;  SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS; |

**3.2 기능별 쿼리**

|  |
| --- |
| **이메일과 패스워드를 입력합니다.** |
| INSERT INTO mydb.users (email, pw)  VALUES ('qwer123@naver.com', '12345'); |
| **회원가입을 완료합니다.** |
| UPDATE mydb.users  SET authentication\_status =1  WHERE email ='qwer123@naver.com'; |
| **사용자가 인증받은 이메일과 패스워드를 입력하여 로그인합니다.** |
| SELECT \*  FROM mydb.users  WHERE email ='qwer123@naver.com'  AND pw ='12345' AND authentication\_status =1; |
| **마이페이지에서는 닉네임을 설정합니다.** |
| Update users  Sent ickname ='asdf'  Where users.email='aaa@gmail.com'; |
| **마이페이지에서는 내가 평가한 작품을 확인할 수 있습니다.** |
| Select collection\_title  From collections  Left join users  On collections.users\_user\_id =users.user\_id  Where nickname ='bbb' and users.user\_id =13; |

|  |
| --- |
| **마이페이지에서는 다른 사용자를 팔로우 할 수 있습니다** |
| Insert into follow (users\_user\_id, Users\_user\_id1) values(  (6 #내 유저 아이디  (select user\_id  From users  Where email ='bbb@gmail.com') #팔로우할 사람의 유저 아이디  ); |
| **마이페이지에서는 나를 팔로우하는 회원 목록을 확인 할 수 있습니다** |
| Select nickname, email  From users  left join follow  on users.user\_id =follow.users\_user\_id  where user\_id = 6; |
| **마이페이지에서는 내가 팔로잉한 회원 목록을 확인 할 수 있습니다.** |
| Select nickname, email  From users  left join follow  on users.user\_id =follow.users\_user\_id  where nickname ='bbb' and email ='bbb@gmail.com'; |
| **작품에 대한 코멘트와 별점을 남길 수 있습니다.(수정 가능)** |
| 별점이나 코멘트 남기기  INSERT INTO mydb.rate (users\_user\_id,Content\_content\_id,rate)  VALUES(6, 6, 3)  INSERT INTO mydb.comment (users\_user\_id,Content\_content\_id,comment\_content)  VALUES(6, 6, '작화가 너무 멋져요')  수정하기 –  Update mydb.rate set rate ='5' whereusers\_user\_id =20;  Update mydb.comment set comment\_content ='끝까지 봐야하는 명작!'  where users\_user\_id =20; |

|  |
| --- |
| **욕설/비방어는 사용할 수 없습니다.** |
| SELECT comment\_content FROM comment; |
| **전체 작품의 전체 평균 별점을 확인할 수 있습니다.** |
| Select content\_name, round(avg(rate),1)  From mydb.rate  join mydb.Content  on mydb.rate.Content\_content\_id =mydb.Content.content\_id  where Content\_content\_id =6; |
| **작품의 평가 상태를 확인할 수 있습니다.** |
| SELECTcontent\_name,  CASE  WHEN mydb.rate.rate ISNULL THEN '미평가'  ELSE mydb.rate.rate  END AS '평가상태'  FROM mydb.Content  LEFT JOIN mydb.rate ON mydb.Content.content\_id =mydb.rate.Content\_content\_id  WHERE users\_user\_id =6; |

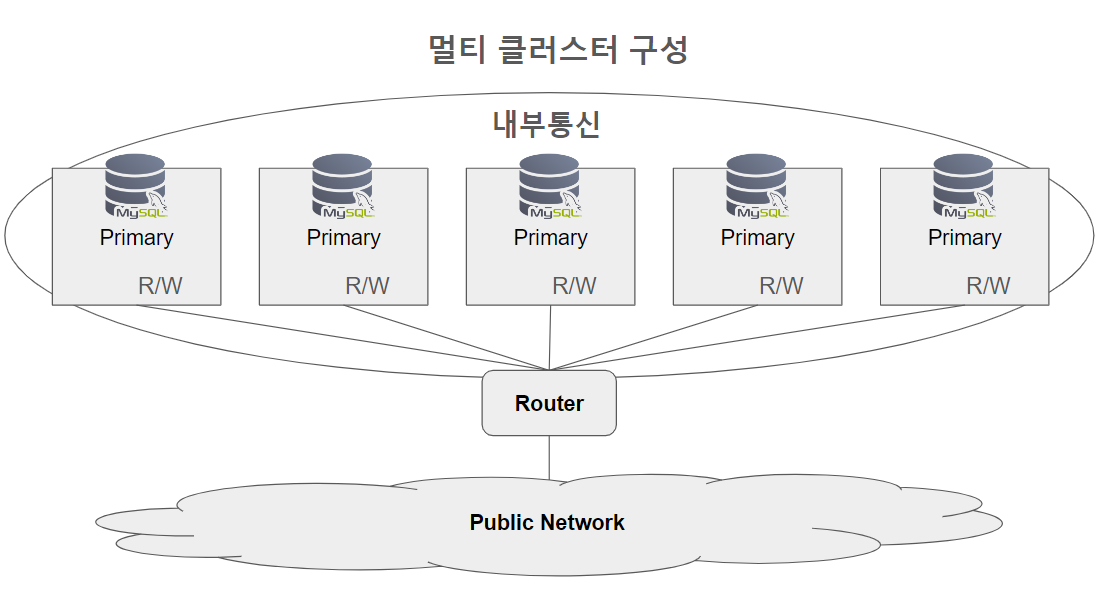
|  |
| --- |
| **작품명, 작가를 검색하여 작품을 찾을 수 있습니다.** |
| 뷰 생성  Use mydb;  CREATE VIEW search as  Select Content.content\_name, categories.category\_name, authors.author\_name  From mydb.manage\_category  join mydb.categories  on manage\_category.categories\_category\_id =categories.category\_id  join mydb.Content  on manage\_category.Content\_content\_id =Content.content\_id  join mydb.own  on manage\_category.Content\_content\_id =own.Content\_Content\_id  join mydb.authors  on own.authors\_author\_id =authors.author\_id;  작품명으로 검색  Select \*  From search  Where content\_name ='화산귀환';  작가이름으로 검색  Select \*  From search  Where author\_name ='조석'; |
| **라이브톡에는 회원들만 참여가 가능합니다.** |
| SELECT \*  FROM users  WHERE email IS NOT NULL AND authentication\_status = 1; |

|  |
| --- |
| **12시가 되면 이전의 채팅들이 사라집니다.** |
| 12시가 되면 이전의 채팅들이 사라집니다.  # 스케줄 상태 확인  Show global variables like'%schedule%';  # OFF 라면 ON  Set global event\_scheduler=On;  Se t@@global.event\_scheduler=On;  # 진행중인 스케줄 목록  SELECT \* FROMi nformation\_schema.events;  # 초기화  Drop event ifexists check\_expected\_end\_date;  # 실행  Create event check\_expected\_end\_date  # 하루마다  On schedule every 1day  # 기준시간  starts '2023-11-01 12:00:00'  enable  do  DELETE FROM Livetalks; |

|  |
| --- |
| **채팅들이 사라지기 전에 사용자들은 채팅 내용을 백업할 수 있습니다.** |
| 채팅 내역 조회  SELECT users.user\_id, users.nickname , content, time FROM Livetalks  JOIN chat  ON Livetalks.livetalk\_id =chat.Livetalks\_livetalk\_id1  JOIN users  ON chat.users\_user\_id1 =users.user\_id;  백업 테이블 3개  mysqldump -u [사용자 계정] -p [패스워드] [원본 데이터베이스명] [테이블 명] >[생성할 백업 DB명].sql  mysqldump -u test\_user -p Livetalks chat users >backup\_db.sql  ~# 비번 : |
| **라이브톡에서 욕설/비방어를 입력할 경우 채팅을 보낼 수 없다는 메세지가 뜹니다.** |
| Show global variables like '%schedule%';  Set global event\_scheduler=On;  Set @@global.event\_scheduler=On;  SELECT \* FROMinformation\_schema.events;  Drop event ifexists check\_filter;  Create event check\_filter  On schedule every 1second  starts current\_timestamp  enable  do  SELECT \* FROM FILTER WHERE (SELECT content FROM Livetalks)  LIKE '%'OR slang\_content OR '%'; |
| **라이브톡은 상시로 참여할 수 있습니다.** |
| SELECT \* FROM Livetalks; |

|  |
| --- |
| **사용자는 작품들을 담아서 컬렉션들을 만들 수 있습니다.** |
| insert into mydb.collections (collection\_title, users\_user\_id)  value ('강추 웹소설', 20);  Insert i manage\_content (collections\_collection\_id1, Content\_content\_id)  Values ( 20, 13 ) ; |
| **회원 닉네임, 컬렉션 이름을 검색하여 다른 사용자가 만든 컬렉션을 볼 수 있습니다.** |
| # 닉네임으로 컬렉션 조회  Select mydb.collections.collection\_title, mydb.Content.content\_name  From collections  join manage\_content  on mydb.collections.collection\_id =mydb.manage\_content.collections\_collection\_id1  join Content  on manage\_content.Content\_content\_id =mydb.Content.content\_id  join users  on mydb.collections.users\_user\_id =mydb.users.user\_id  where nickname='bbb';  # 작품명으로 컬렉션 조회  Select mydb.users.nickname, mydb.collections.collection\_title, mydb.Content.content\_name  From collections  join manage\_content  on mydb.collections.collection\_id =mydb.manage\_content.collections\_collection\_id1  join Content  on manage\_content.Content\_content\_id =mydb.Content.content\_id  join users  on mydb.collections.users\_user\_id =mydb.users.user\_id  where mydb.Content.content\_name like '%아홉%'; |

**3-3. 시스템 아키텍쳐**

****

저희 서비스는 리뷰 작성과 리스트 만들기 그리고 채팅 등의 서비스가 활발하게 이용될 수 있도록 운영할 계획입니다

이러한 서비스를 운영하게 되면 읽기와 쓰기 기능을 모두 자주 사용해야 될것이라 예상됩니다

그래서 비용 절감에도 유리하고 부하 분산과 높은 가용성으로 인해 안정적인 서버 상태를 유지할 수 있는 멀티 클러스터 방식으로 서버를 구성했습니다

[서버 구성]

PC 6대 준비

> PC 5대를 클러스터 노드로 만들어 DB 서버를 구성

> PC 1대는 라우터로 구성해서 라우터를 통해 DB에 접근

\*6646 포트는 읽기/쓰기 포트이다. (9999 포트로 포트 포워딩)