University of Dayton

CPS 542: Database Systems Management I Final Report

Calendar-based Social Media Database

By Vincent Durham

August 4th, 2023

Table of Contents

Description of Database

Introduction:

There are many different social media apps in the world today, and most are used to share recent experiences with friends and family. People have hundreds of pictures saved in their phones that they enjoy posting. Most modern social media apps rely on users to post in real-time, however not many apps use preset posts. My thought to fix this would be a calendar social media app, with preset calendars made with pictures and quotes(captions) for a full year. Users could follow other users to view their calendars, but also have private calendars between just family and friends. Each calendar would be composed of pictures and quotes, either randomly inserted or set to a specific date. Users would have followers just like most other social media, and settings for specific followers could be adjusted.

Objective:

The main objective of this project is to create a database management system for a social media app that allows users to build and share calendars with friends. Users will create multiple calendars and share them with their followers. To create a calendar would cost money, so billing Users will be used to make money from the calendars. The objective of this database will be to have Users who can follow other users, create calendars using pictures and quotes, and be billed for their usage. Also, a User can choose their settings based on if they would like their made calendars to be public or private, and if they want to use location services.

Entities

- ***User -*** The User entity will contain the login information (user_id, username password, phone_number) and the data belonging to the User (calendars_viewable, calendars_created, followers_list, billing_information)
- ***Follower -*** Another User entity that is following a User. (This will be a user entity, but also include date_followed)
- ***Calendar -*** A 365 day calendar that would be made up of pictures and quotes. It would know which day to display and which user created the calendar. (date_created)
- ***Pictures -*** The set of pictures belonging to a certain calendar. User could choose specific pictures for specific days, or use random insertions (picture_id, picture_url, selected_day)
- ***Quotes -*** The set of quotes belonging to a certain calendar. User could choose specific quotes for specific days, or use random insertions (quote_id, quote, selected_day)

- ***Billing -*** Contains the money information of the User (billing_id, total_spent, card_number, payment_method)
- ***Settings -*** Built-in user settings that a user can choose from for their app experience

Relationships

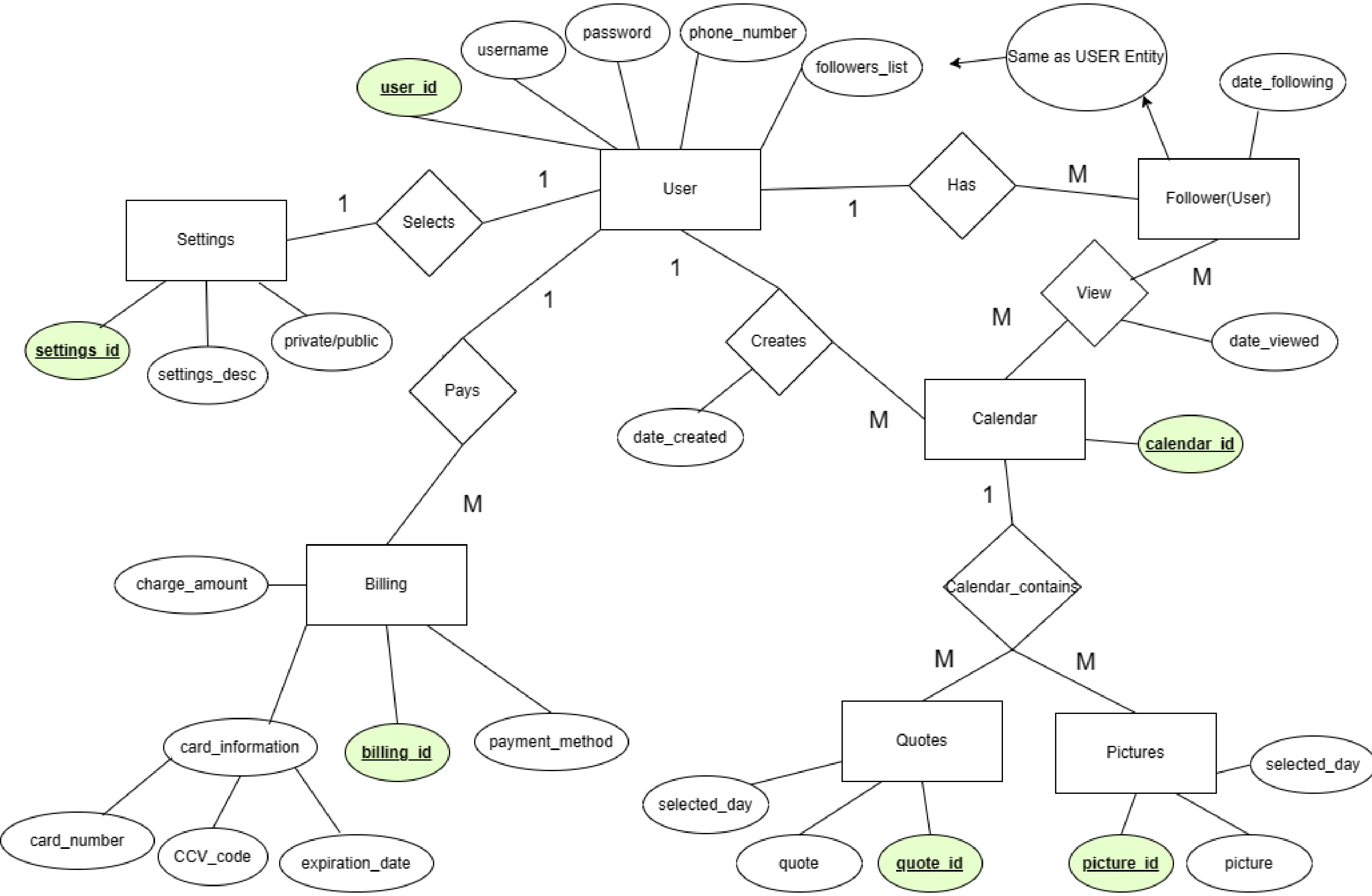
- ***Pays:*** *User* ***pays*** *Billing*
- ***Contains:*** *Calendar* ***contains*** *Pictures & Quotes*
- ***Views:*** *User* ***views*** *Follower's Calendar*
- ***Creates:*** *User* ***creates*** *a Calendar*
- ***Follows:*** *User* ***follows*** *another User*
- ***Displays:*** *Calendar* ***displays*** *a Picture & Quote*
- ***Selects****: User* ***selects*** *Settings*

Transactions

1. Report total number of calendars and followers for a User
2. Update the Pictures or Quotes
3. Insert a new calendar for a User
4. Report the top 3 highest spending Users
5. Report the total spent by all customers
6. Update a calendar from public or private
    a. Update the followers who can view a private calendar
7. Insert a new follower for a User
8. Remove a calendar for a User
9. Update the Settings for a User

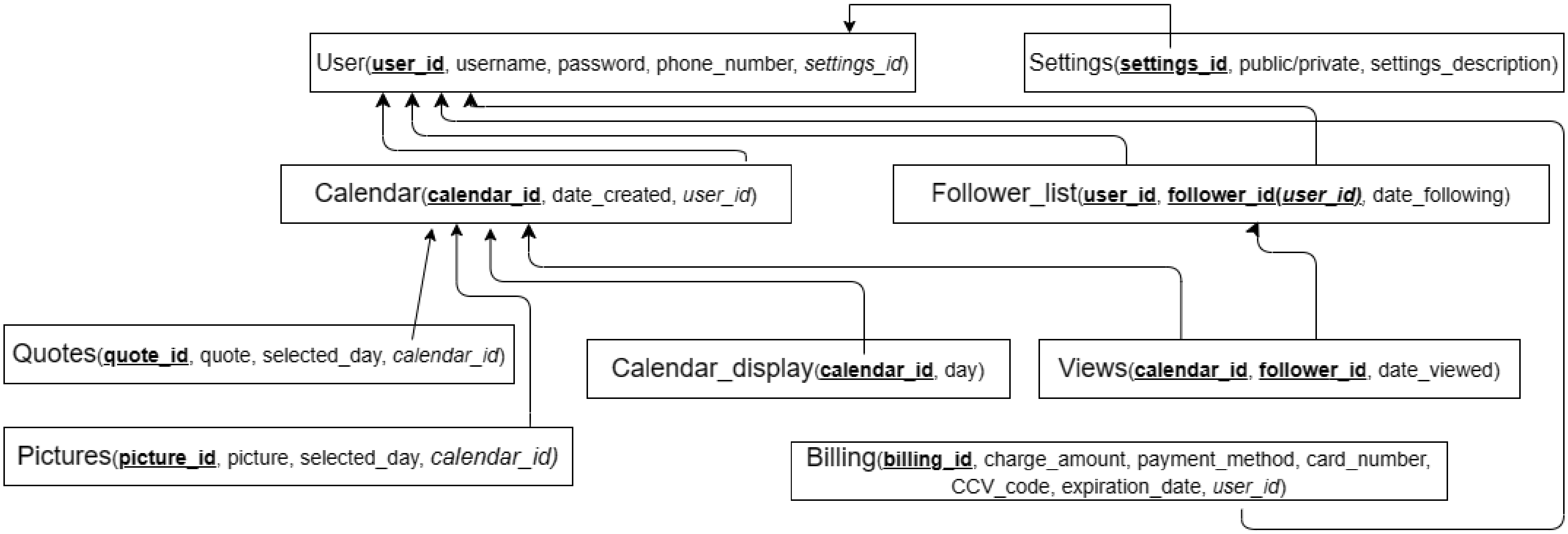
ER Diagram

An Entity-Relationship Diagram of the Database.



Relational Schema

Relational Schema for Calendar Social Media App


User(user_id, username, password, phone_number, settings_id)
Settings(settings_id, public/private, settings_description)
Calendar(calendar_id, date_created, user_id)
Follower_list(user_id, follower_id(user_id), date_following)
Quotes(quote_id, quote, selected_day, calendar_id)
Calendar_display(calendar_id, day)
Views(calendar_id, follower_id, date_viewed)
Pictures(picture_id, picture, selected_day, calendar_id)
Billing(billing_id, charge_amount, payment_method, card_number, CCV_code, expiration_date, user_id)

Normalization

Normalization for Calendar Social Media App

1NF

- A relation is in First Normal Form if every attribute is single-valued for each cell and the domains of the attributes are atomic, meaning no repeating fields in domains.

In my database, there were multiple 1NF relations that needed to be broken down. In User, followers_list was made because if it remained an attribute, it could have a big number of values. So, it is broken into a separate table where two user_ids act as the primary key. The first user id is being followed and the second user id is the follower, and this is kept with an attribute of the date when the following happened. Also, card information has already been broken into card_number, CCV_code, and expiration_date, so each is a single value in Billing.

User (**user_id**, username, password, phone_number, *settings_id*)

Followers_list(**user_id**, **user_id**, date_following)

Calendar (**calendar_id**, date_created, *user_id*)

Billing (**billing_id**, charge_amount, card_number, CCV_code, expiration_date,

payment_method, *user_id*)

Quotes (**quote_id**, quote, selected_day, *calendar_id*)

Pictures (**picture_id**, picture, selected_day, *calendar_id*)

Creates (**user_id**, **calendar_id**, date_created)

Settings(**settings_id**, public/private, settings_description)

Views (**follower_id**, **calendar_id**, date_viewed)

Calendar_display(**calendar_id**, day)

2NF

- A relation is in Second Normal Form if the schema satisfies 1NF and all of the non-key attributes are fully functionally dependent on the key.

All of these functions are in 2NF

User (**user_id**, username, password, phone_number, *settings_id*)

Followers_list(**user_id**, **user_id**, date_following)

Calendar (**calendar_id**, date_created, *user_id*)

Billing (**billing_id**, charge_amount, card_number, CCV_code, expiration_date,

payment_method, *user_id*)

Quotes (**quote_id**, quote, selected_day, *calendar_id*)

Pictures (**picture_id**, picture, selected_day, *calendar_id*)

Creates (**user_id**, **calendar_id**, date_created)

Settings(**settings_id**, public/private, settings_description)

Views (**follower_id**, **calendar_id**, date_viewed)

Calendar_display(**calendar_id**, day)

3NF

- A relation is in Third Normal Form if the schema satisfies 2NF and has no transitive dependencies.

All of these functions are in 3NF

User (**user_id**, username, password, phone_number, *settings_id*)

Followers_list(**user_id**, **user_id**, date_following)

Calendar (**calendar_id**, date_created, *user_id*)

Billing (**billing_id**, charge_amount, card_number, CCV_code, expiration_date, payment_method, *user_id*)

Quotes (**quote_id**, quote, selected_day, *calendar_id*)

Pictures (**picture_id**, picture, selected_day, *calendar_id*)

Creates (**user_id**, **calendar_id**, date_created)

Settings(**settings_id**, public/private, settings_description)

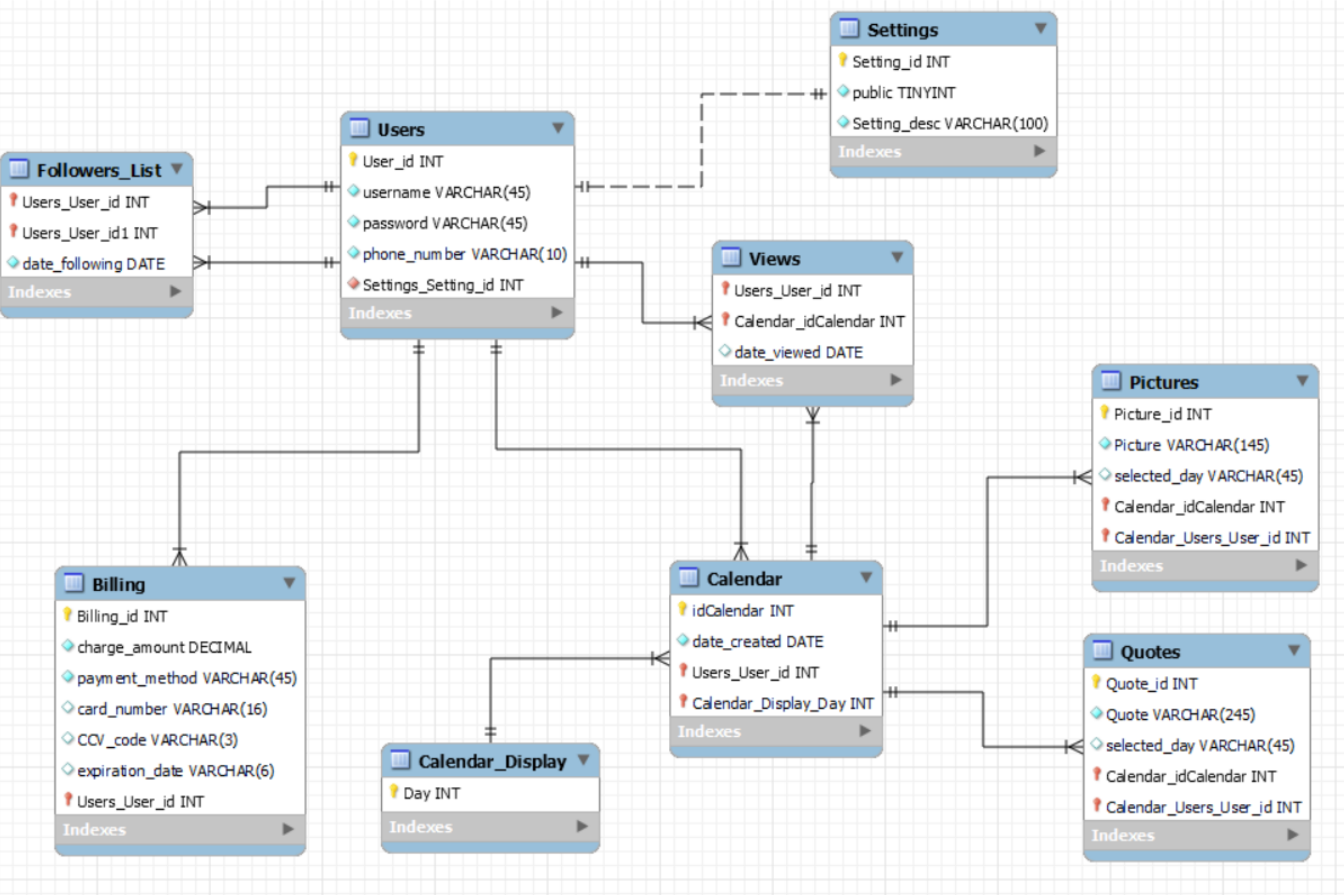
Views (**follower_id**, **calendar_id**, date_viewed)

Calendar_display(**calendar_id**, day)

Normalization was hard for this project once I pivoted my database because the data I was attempting to add didn’t make sense with a Follower entity. This caused me to build a normalized relational schema to be sure it would work, so most of the entities were broken down into single-value attributes beforehand.

Logical Model

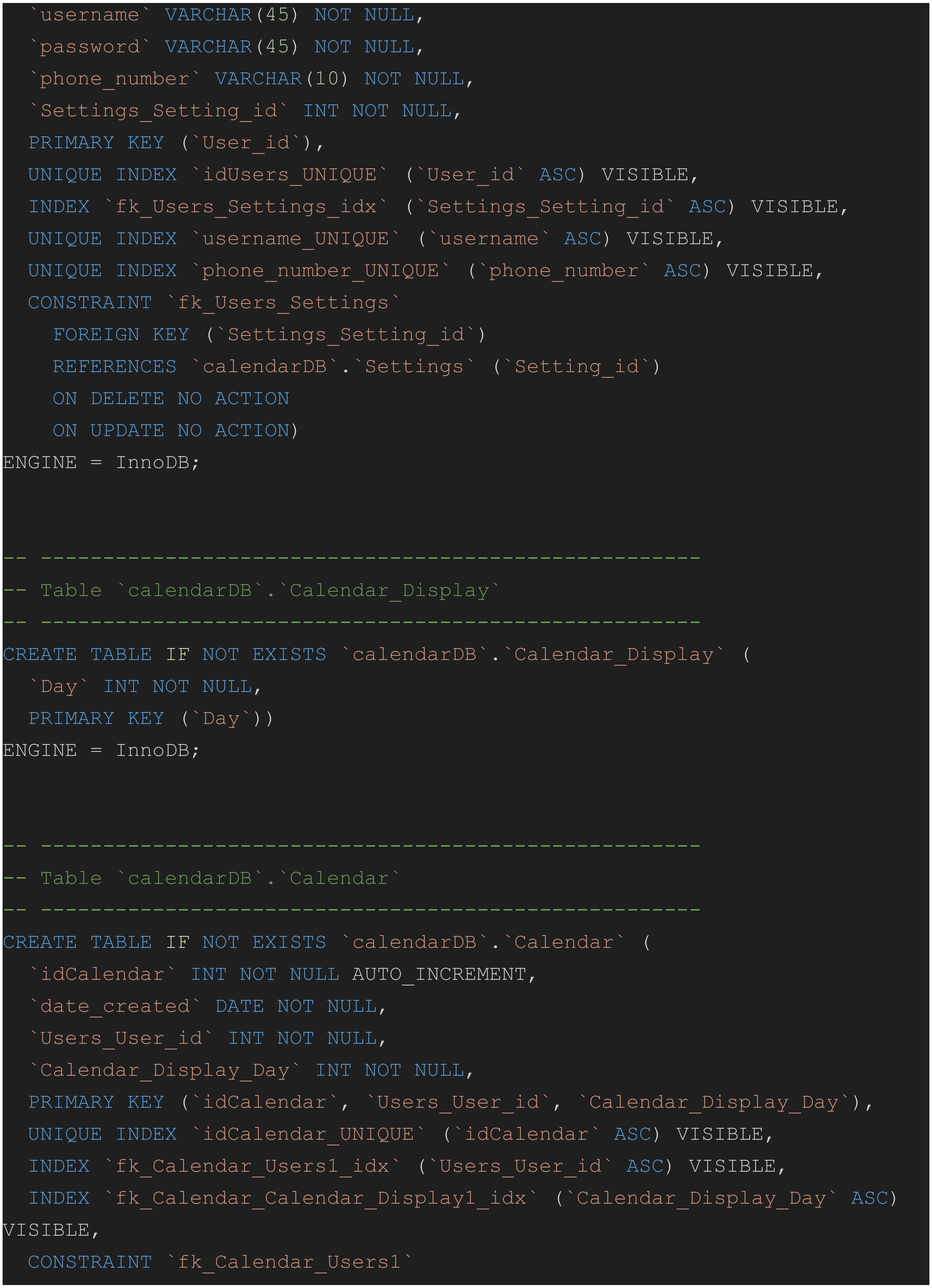
The logical model created in MySQL Workbench to create the schema for our database.



Physical Model

Here is the physical SQL code that was forward engineered by MySQL WorkBench to create a working database out of our provided Logical Model.

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema calendarDB
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema calendarDB
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `calendarDB` DEFAULT CHARACTER SET utf8 ;
USE `calendarDB` ;

-- -----------------------------------------------------
-- Table `calendarDB`.`Settings`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Settings` (
  `Setting_id` INT NOT NULL AUTO_INCREMENT,
  `public` TINYINT NOT NULL,
  `Setting_desc` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`Setting_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Users`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Users` (
  `User_id` INT NOT NULL AUTO_INCREMENT,
```



```
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `phone_number` VARCHAR(10) NOT NULL,
  `Settings_Setting_id` INT NOT NULL,
  PRIMARY KEY (`User_id`),
  UNIQUE INDEX `idUsers_UNIQUE` (`User_id` ASC) VISIBLE,
  INDEX `fk_Users_Settings_idx` (`Settings_Setting_id` ASC) VISIBLE,
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE,
  UNIQUE INDEX `phone_number_UNIQUE` (`phone_number` ASC) VISIBLE,
  CONSTRAINT `fk_Users_Settings`
    FOREIGN KEY (`Settings_Setting_id`)
    REFERENCES `calendarDB`.`Settings` (`Setting_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Calendar_Display`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Calendar_Display` (
  `Day` INT NOT NULL,
  PRIMARY KEY (`Day`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Calendar`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Calendar` (
  `idCalendar` INT NOT NULL AUTO_INCREMENT,
  `date_created` DATE NOT NULL,
  `Users_User_id` INT NOT NULL,
  `Calendar_Display_Day` INT NOT NULL,
  PRIMARY KEY (`idCalendar`, `Users_User_id`, `Calendar_Display_Day`),
  UNIQUE INDEX `idCalendar_UNIQUE` (`idCalendar` ASC) VISIBLE,
  INDEX `fk_Calendar_Users1_idx` (`Users_User_id` ASC) VISIBLE,
  INDEX `fk_Calendar_Calendar_Display1_idx` (`Calendar_Display_Day` ASC)
VISIBLE,
  CONSTRAINT `fk_Calendar_Users1`
```

```
    FOREIGN KEY (`Users_User_id`)
    REFERENCES `calendarDB`.`Users` (`User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Calendar_Calendar_Display1`
    FOREIGN KEY (`Calendar_Display_Day`)
    REFERENCES `calendarDB`.`Calendar_Display` (`Day`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Quotes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Quotes` (
  `Quote_id` INT NOT NULL AUTO_INCREMENT,
  `Quote` VARCHAR(245) NOT NULL,
  `selected_day` VARCHAR(45) NULL,
  `Calendar_idCalendar` INT NOT NULL,
  `Calendar_Users_User_id` INT NOT NULL,
  PRIMARY KEY (`Quote_id`, `Calendar_idCalendar`,
`Calendar_Users_User_id`),
  INDEX `fk_Quotes_Calendar1_idx` (`Calendar_idCalendar` ASC,
`Calendar_Users_User_id` ASC) VISIBLE,
  CONSTRAINT `fk_Quotes_Calendar1`
    FOREIGN KEY (`Calendar_idCalendar` , `Calendar_Users_User_id`)
    REFERENCES `calendarDB`.`Calendar` (`idCalendar` , `Users_User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Pictures`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Pictures` (
  `Picture_id` INT NOT NULL AUTO_INCREMENT,
  `Picture` VARCHAR(145) NOT NULL,
  `selected_day` VARCHAR(45) NULL,
  `Calendar_idCalendar` INT NOT NULL,
  `Calendar_Users_User_id` INT NOT NULL,
  PRIMARY KEY (`Picture_id`, `Calendar_idCalendar`,
`Calendar_Users_User_id`),
  INDEX `fk_Pictures_Calendar1_idx` (`Calendar_idCalendar` ASC,
`Calendar_Users_User_id` ASC) VISIBLE,
  CONSTRAINT `fk_Pictures_Calendar1`
    FOREIGN KEY (`Calendar_idCalendar` , `Calendar_Users_User_id`)
    REFERENCES `calendarDB`.`Calendar` (`idCalendar` , `Users_User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Billing`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Billing` (
  `Billing_id` INT NOT NULL AUTO_INCREMENT,
  `charge_amount` DECIMAL NOT NULL,
  `payment_method` VARCHAR(45) NOT NULL,
  `card_number` VARCHAR(16) NULL,
  `CCV_code` VARCHAR(3) NULL,
  `expiration_date` VARCHAR(6) NULL,
  `Users_User_id` INT NOT NULL,
  PRIMARY KEY (`Billing_id`, `Users_User_id`),
  INDEX `fk_Billing_Users1_idx` (`Users_User_id` ASC) VISIBLE,
  CONSTRAINT `fk_Billing_Users1`
    FOREIGN KEY (`Users_User_id`)
    REFERENCES `calendarDB`.`Users` (`User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Views`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Views` (
  `Users_User_id` INT NOT NULL,
  `Calendar_idCalendar` INT NOT NULL,
  `date_viewed` DATE NULL,
  PRIMARY KEY (`Users_User_id`, `Calendar_idCalendar`),
  INDEX `fk_Users_has_Calendar_Calendar1_idx` (`Calendar_idCalendar` ASC) VISIBLE,
  INDEX `fk_Users_has_Calendar_Users1_idx` (`Users_User_id` ASC) VISIBLE,
  CONSTRAINT `fk_Users_has_Calendar_Users1`
    FOREIGN KEY (`Users_User_id`)
    REFERENCES `calendarDB`.`Users` (`User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Users_has_Calendar_Calendar1`
    FOREIGN KEY (`Calendar_idCalendar`)
    REFERENCES `calendarDB`.`Calendar` (`idCalendar`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `calendarDB`.`Followers_List`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `calendarDB`.`Followers_List` (
  `Users_User_id` INT NOT NULL,
  `Users_User_id1` INT NOT NULL,
  `date_following` DATE NOT NULL,
  PRIMARY KEY (`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 `calendarDB`.`Users` (`User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Users_has_Users_Users2`
    FOREIGN KEY (`Users_User_id1`)
    REFERENCES `calendarDB`.`Users` (`User_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

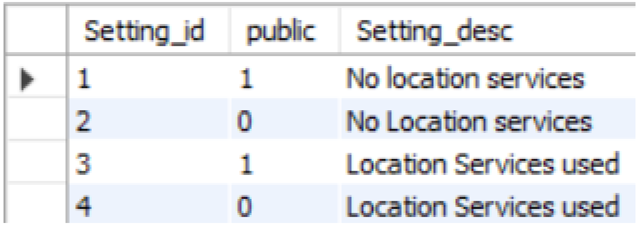
```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Inserted Data and Tables

Provided next are the Tables and the data inserted into them once the database was created through the forward engineering. The Settings Table had to be filled before the User Table because of a foreign key constraint, as well as the selected day to display table being filled before the Calendar table had data inserted into it.

Settings Table

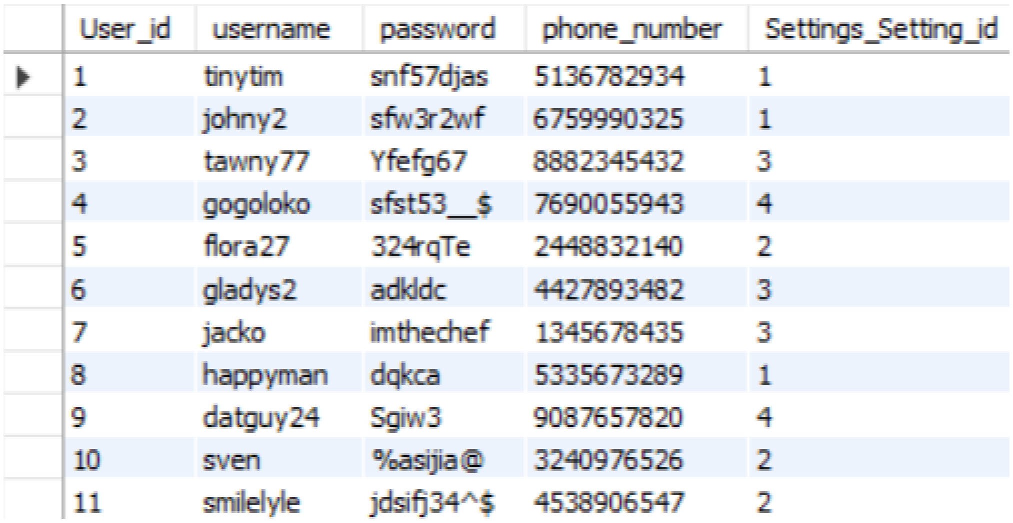
The Settings table could be improved but for now, a user chooses one of 4 preset settings, either making their profile public or private and allowing location services. This table could be updated by the user through a settings screen.



	Setting_id	public	Setting_desc
▶	1	1	No location services
	2	0	No Location services
	3	1	Location Services used
	4	0	Location Services used

User Table

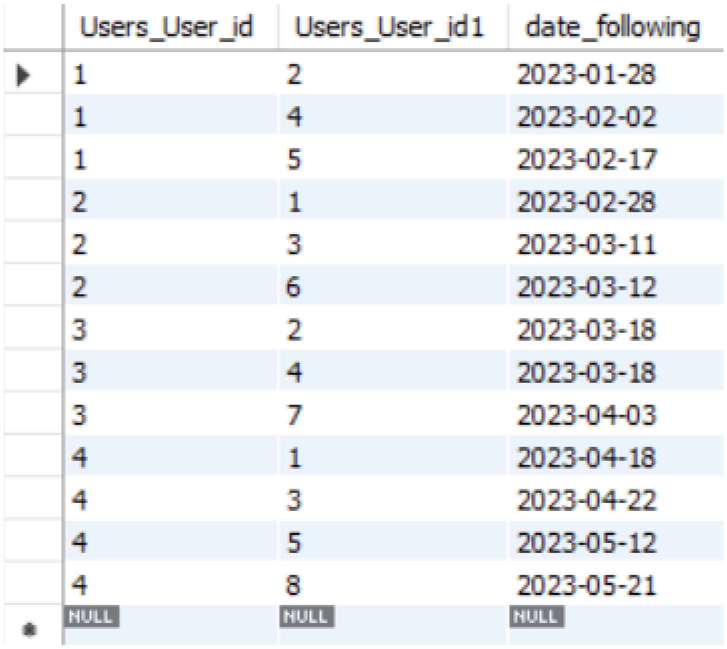
The User Table contains all of the users and their login credentials. It also contains the preset settings a user selects



	User_id	username	password	phone_number	Settings_Setting_id
▶	1	tinytim	snf57djas	5136782934	1
	2	johny2	sfw3r2wf	6759990325	1
	3	tawny77	Yfefg67	8882345432	3
	4	gogoloko	sfst53__$	7690055943	4
	5	flora27	324rqTe	2448832140	2
	6	gladys2	adkldc	4427893482	3
	7	jacko	imthechef	1345678435	3
	8	happyman	dqkca	5335673289	1
	9	datguy24	Sgiw3	9087657820	4
	10	sven	%asijia@	3240976526	2
	11	smilelyle	jdsifj34^$	4538906547	2

Followers Table

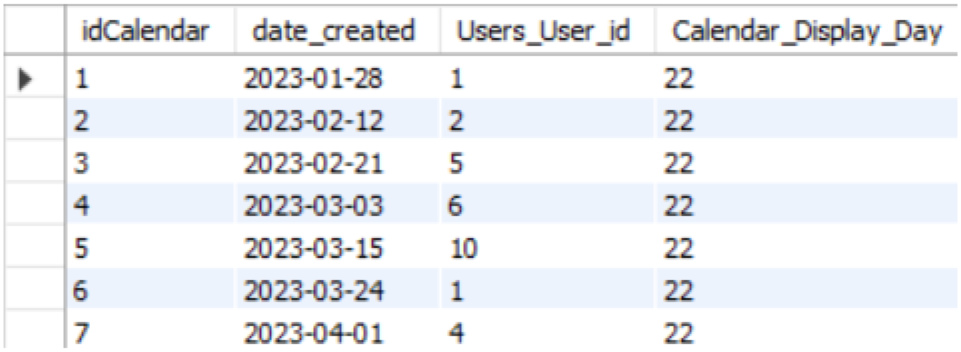
The Followers Table shows two different user ids. The first is the id of the user being followed. The second is the id of the user following the first user. Finally, it keeps track of the date a user first follows another user.



	Users_User_id	Users_User_id1	date_following
▶	1	2	2023-01-28
	1	4	2023-02-02
	1	5	2023-02-17
	2	1	2023-02-28
	2	3	2023-03-11
	2	6	2023-03-12
	3	2	2023-03-18
	3	4	2023-03-18
	3	7	2023-04-03
	4	1	2023-04-18
	4	3	2023-04-22
	4	5	2023-05-12
	4	8	2023-05-21
*	NULL	NULL	NULL

<u>Calendar Table</u>

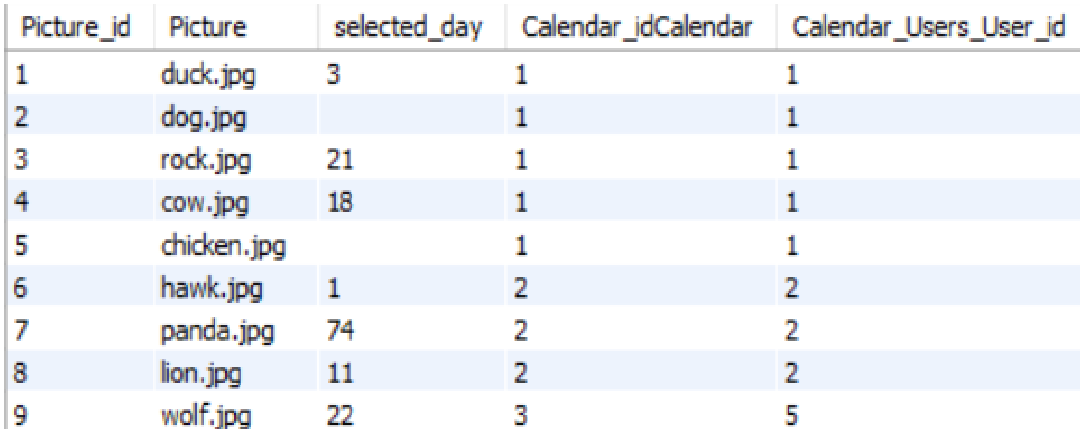
The Calendar Table shows each calendar made, the date it was created and the user that created that calendar. The Calendar Display day will always be the same number for each calendar but changes daily (1-365).



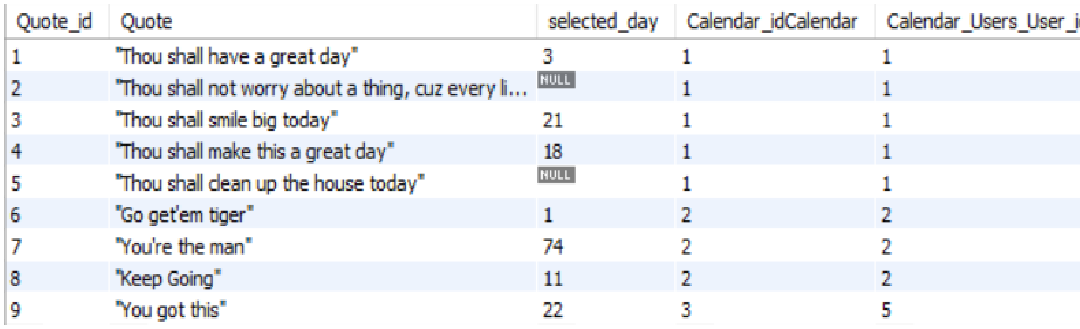
	idCalendar	date_created	Users_User_id	Calendar_Display_Day
▶	1	2023-01-28	1	22
	2	2023-02-12	2	22
	3	2023-02-21	5	22
	4	2023-03-03	6	22
	5	2023-03-15	10	22
	6	2023-03-24	1	22
	7	2023-04-01	4	22

<u>Picture Table & Quote Table</u>

The Picture and Quote Tables hold all of the pictures and quotes contained in each calendar. They also can save a selected day for that picture or quote to be posted when the calendar display number matches the selected_day. It contains an extra column that tells you which user created the calendar that the quote or picture is used in.



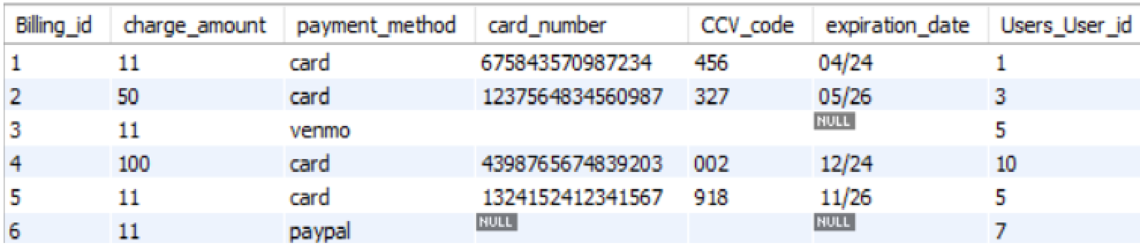
Picture_id	Picture	selected_day	Calendar_idCalendar	Calendar_Users_User_id
1	duck.jpg	3	1	1
2	dog.jpg		1	1
3	rock.jpg	21	1	1
4	cow.jpg	18	1	1
5	chicken.jpg		1	1
6	hawk.jpg	1	2	2
7	panda.jpg	74	2	2
8	lion.jpg	11	2	2
9	wolf.jpg	22	3	5



Quote_id	Quote	selected_day	Calendar_idCalendar	Calendar_Users_User_i
1	"Thou shall have a great day"	3	1	1
2	"Thou shall not worry about a thing, cuz every li...	NULL	1	1
3	"Thou shall smile big today"	21	1	1
4	"Thou shall make this a great day"	18	1	1
5	"Thou shall clean up the house today"	NULL	1	1
6	"Go get'em tiger"	1	2	2
7	"You're the man"	74	2	2
8	"Keep Going"	11	2	2
9	"You got this"	22	3	5

<u>Billing Table</u>

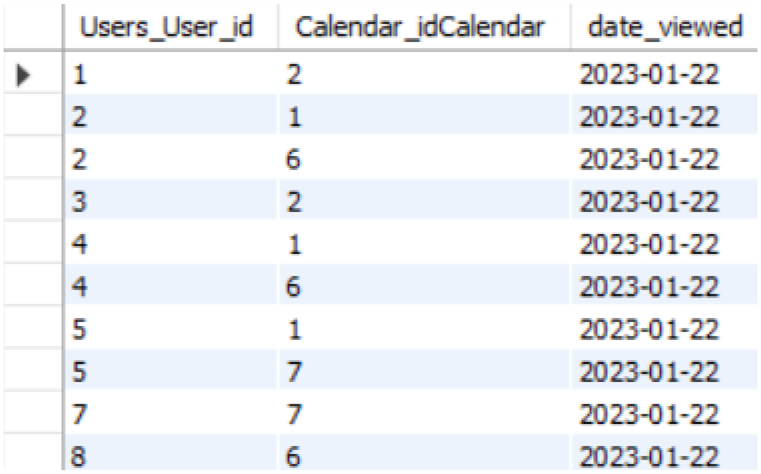
The Billing Table contains the charge amount for each transaction made by a user. If the payment method is card, the card details are taken, if not they are left null.



Billing_id	charge_amount	payment_method	card_number	CCV_code	expiration_date	Users_User_id
1	11	card	675843570987234	456	04/24	1
2	50	card	1237564834560987	327	05/26	3
3	11	venmo			NULL	5
4	100	card	4398765674839203	002	12/24	10
5	11	card	1324152412341567	918	11/26	5
6	11	paypal	NULL		NULL	7

Calendar Views Table

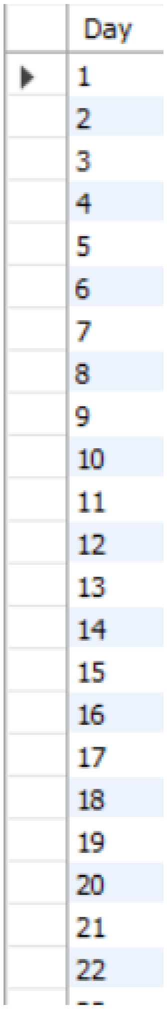
The calendar views table shows which users have viewed which calendar and the date at which they viewed them. Because this is a daily calendar, most view dates would be daily.



Users_User_id	Calendar_idCalendar	date_viewed
1	2	2023-01-22
2	1	2023-01-22
2	6	2023-01-22
3	2	2023-01-22
4	1	2023-01-22
4	6	2023-01-22
5	1	2023-01-22
5	7	2023-01-22
7	7	2023-01-22
8	6	2023-01-22

Selected Day to Display

A trace through 365 days to connect pictures and quotes to their selected days in calendars. This had to be created before the Calendar Table because there is a foreign key in the Calendar Table referencing this table.

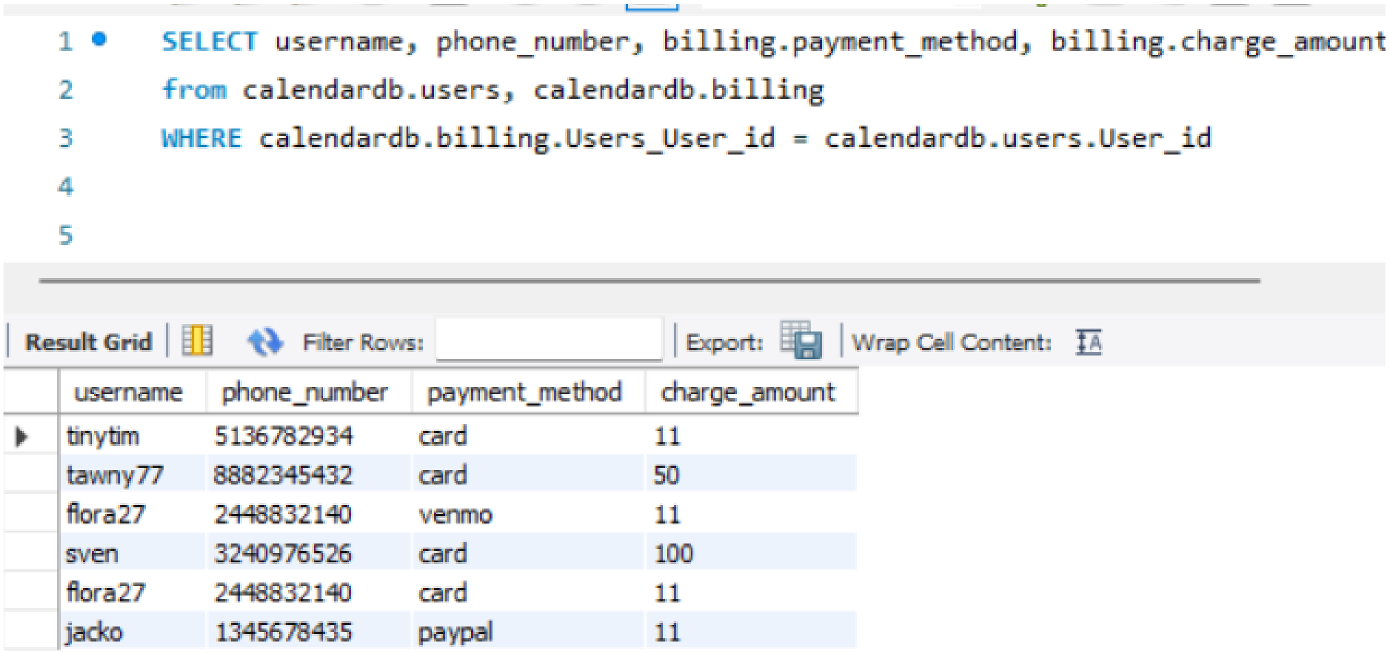


Day
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

SQL Queries

SELECT

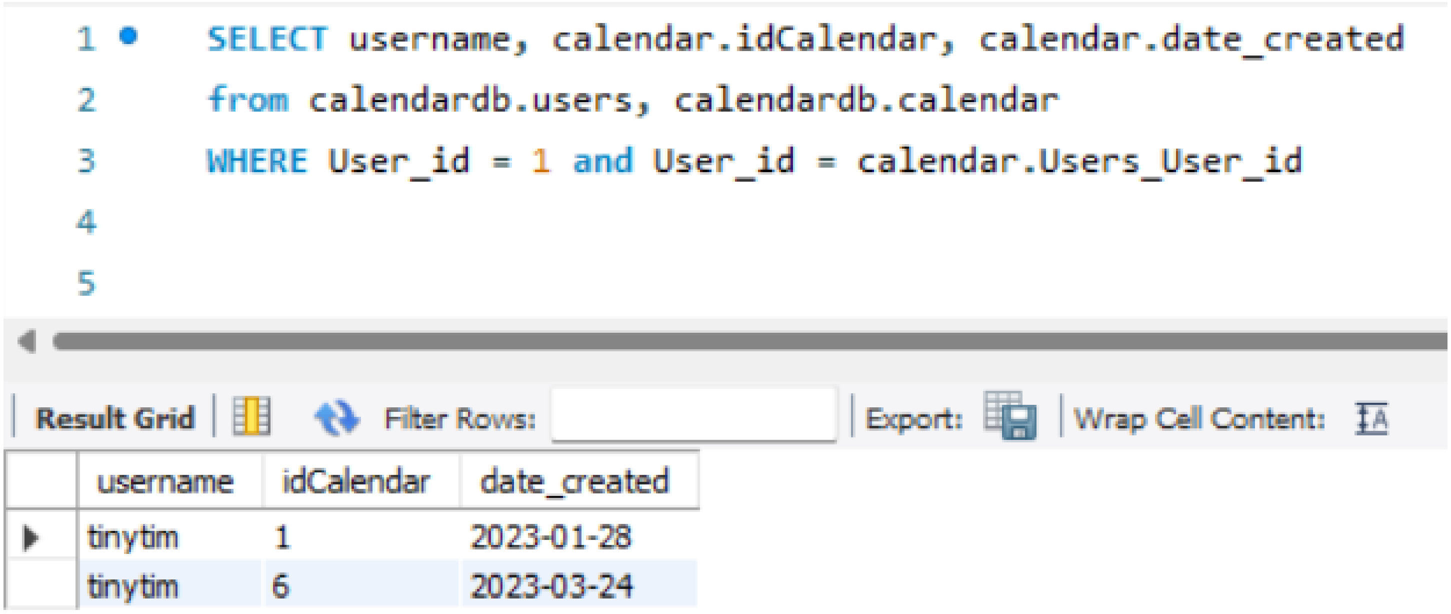
1. Show the username and phone number of each user that has paid a billing statement, also display the charge amount and how the charge was paid.



```
SELECT username, phone_number, billing.payment_method, billing.charge_amount
from calendardb.users, calendardb.billing
WHERE calendardb.billing.Users_User_id = calendardb.users.User_id
```

username	phone_number	payment_method	charge_amount
tinytim	5136782934	card	11
tawny77	8882345432	card	50
flora27	2448832140	venmo	11
sven	3240976526	card	100
flora27	2448832140	card	11
jacko	1345678435	paypal	11

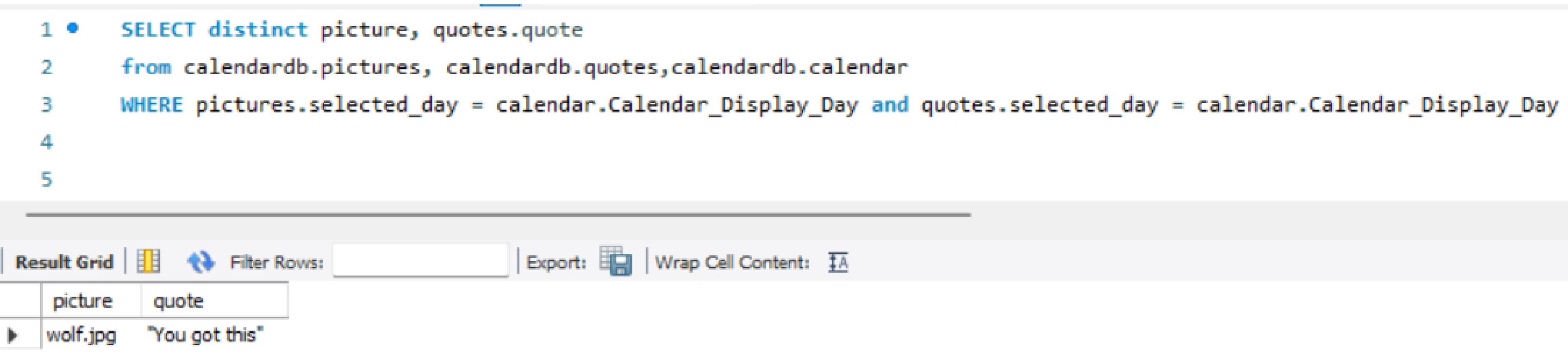
2. Display a user and the calendars they have created.



```
SELECT username, calendar.idCalendar, calendar.date_created
from calendardb.users, calendardb.calendar
WHERE User_id = 1 and User_id = calendar.Users_User_id
```

username	idCalendar	date_created
tinytim	1	2023-01-28
tinytim	6	2023-03-24

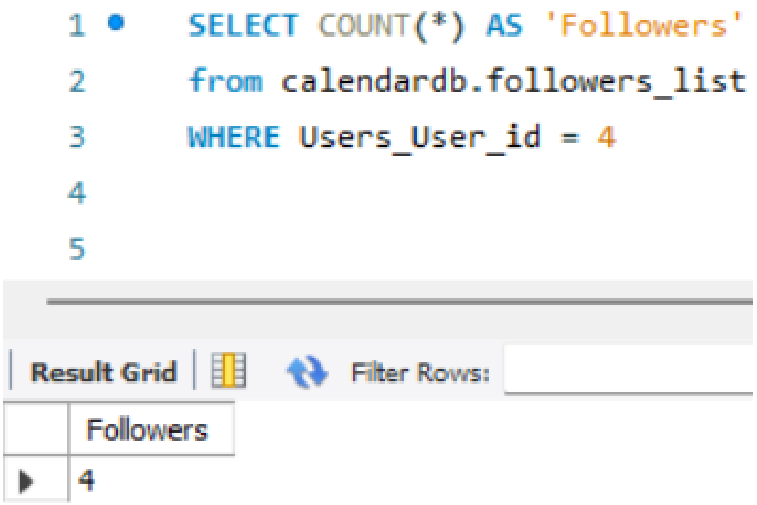
3. Display the picture and quote for a calendar on a selected day



```sql
SELECT distinct picture, quotes.quote
from calendardb.pictures, calendardb.quotes,calendardb.calendar
WHERE pictures.selected_day = calendar.Calendar_Display_Day and quotes.selected_day = calendar.Calendar_Display_Day
```

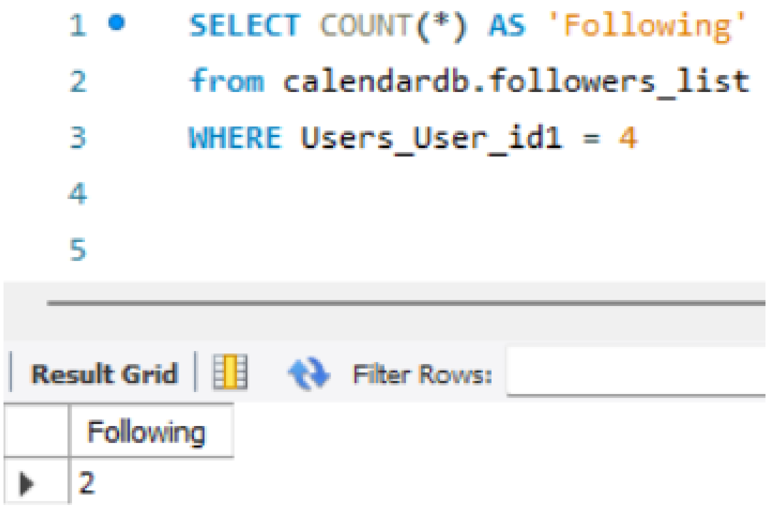
```
   picture    quote
▶  wolf.jpg   "You got this"
```

4. Show how many users a specific user is following, and how many are following them



```sql
SELECT COUNT(*) AS 'Followers'
from calendardb.followers_list
WHERE Users_User_id = 4
```

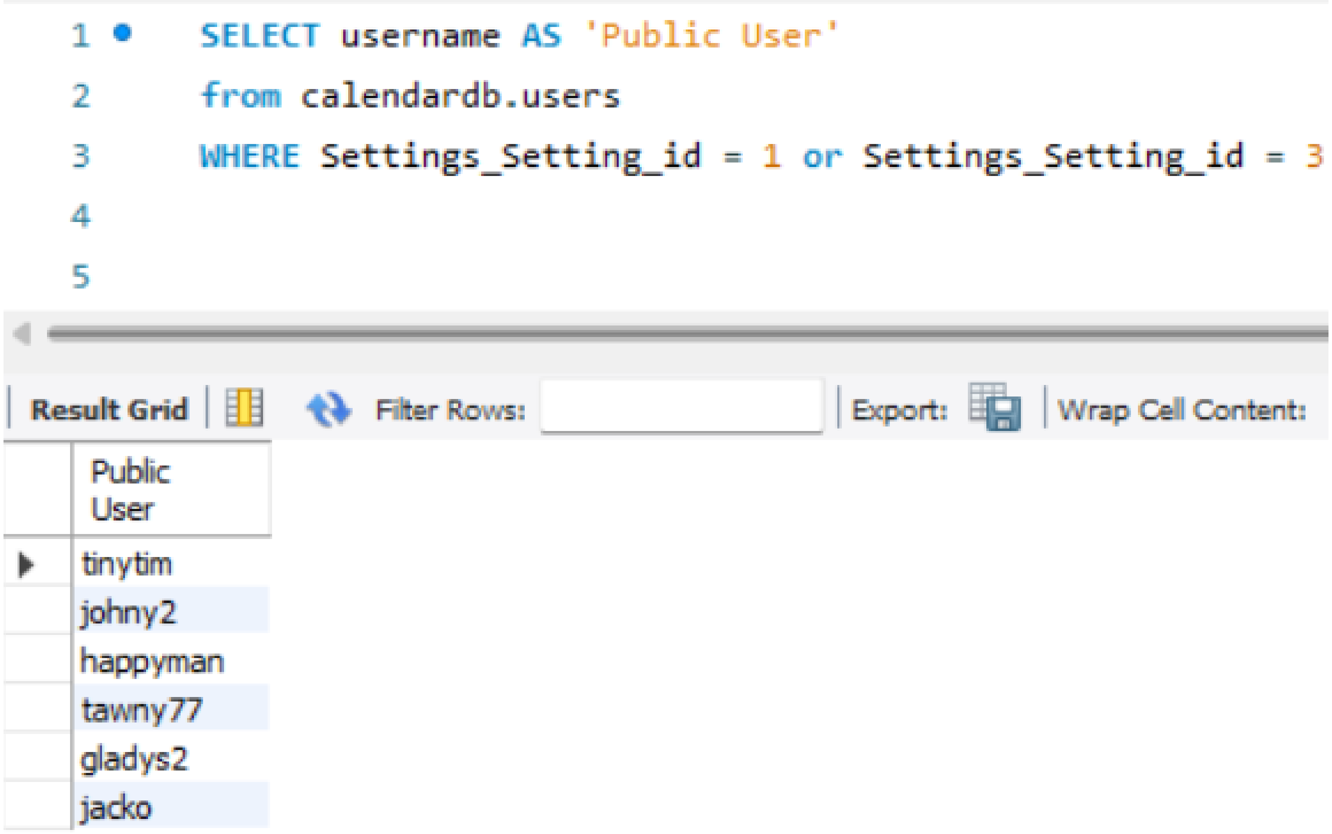
```
   Followers
▶  4
```



```sql
SELECT COUNT(*) AS 'Following'
from calendardb.followers_list
WHERE Users_User_id1 = 4
```

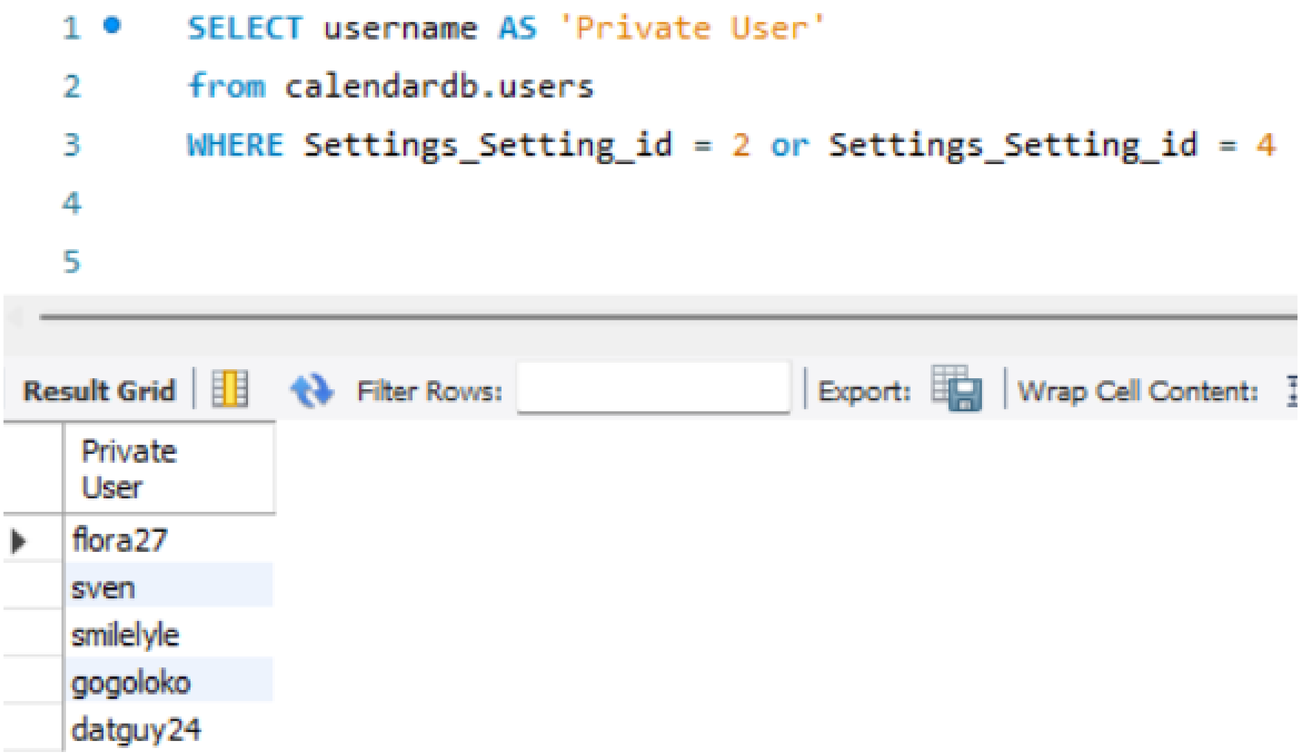
```
   Following
▶  2
```

5. Show which users are public and which users are private



```
SELECT username AS 'Public User'
from calendardb.users
WHERE Settings_Setting_id = 1 or Settings_Setting_id = 3
```

Public User
tinytim
johny2
happyman
tawny77
gladys2
jacko

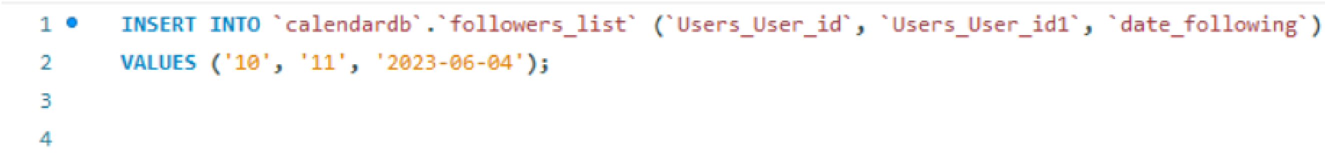


```
SELECT username AS 'Private User'
from calendardb.users
WHERE Settings_Setting_id = 2 or Settings_Setting_id = 4
```

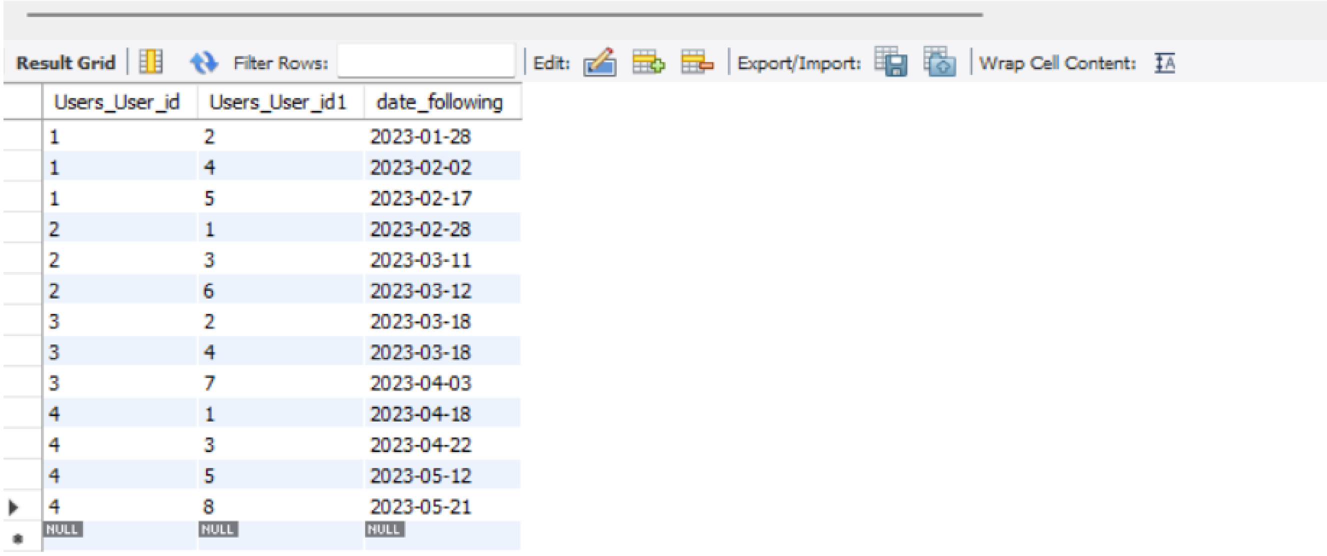
Private User
flora27
sven
smilelyle
gogoloko
datguy24

INSERT

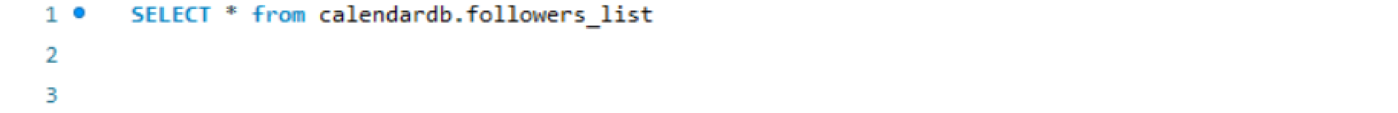
Insert a new follower for user 10(sven). Their new follower is user 11(smilelyle) and they followed on 06/04/2023.



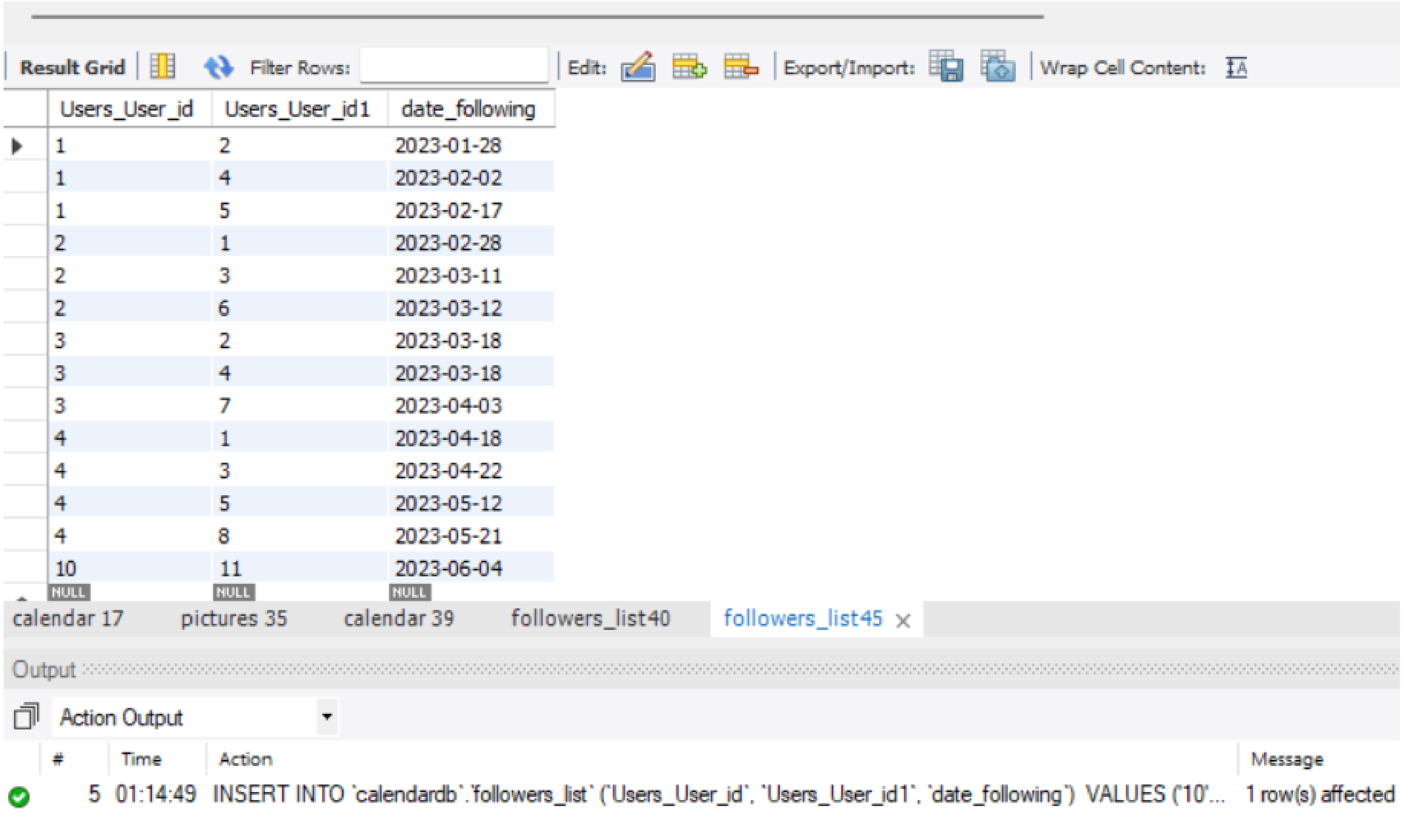
```sql
INSERT INTO `calendardb`.`followers_list` (`Users_User_id`, `Users_User_id1`, `date_following`)
VALUES ('10', '11', '2023-06-04');
```



Users_User_id	Users_User_id1	date_following
1	2	2023-01-28
1	4	2023-02-02
1	5	2023-02-17
2	1	2023-02-28
2	3	2023-03-11
2	6	2023-03-12
3	2	2023-03-18
3	4	2023-03-18
3	7	2023-04-03
4	1	2023-04-18
4	3	2023-04-22
4	5	2023-05-12
4	8	2023-05-21
NULL	NULL	NULL



```sql
SELECT * from calendardb.followers_list
```

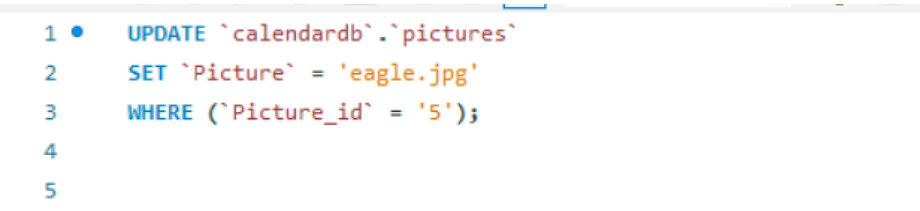


Users_User_id	Users_User_id1	date_following
1	2	2023-01-28
1	4	2023-02-02
1	5	2023-02-17
2	1	2023-02-28
2	3	2023-03-11
2	6	2023-03-12
3	2	2023-03-18
3	4	2023-03-18
3	7	2023-04-03
4	1	2023-04-18
4	3	2023-04-22
4	5	2023-05-12
4	8	2023-05-21
10	11	2023-06-04
NULL	NULL	NULL

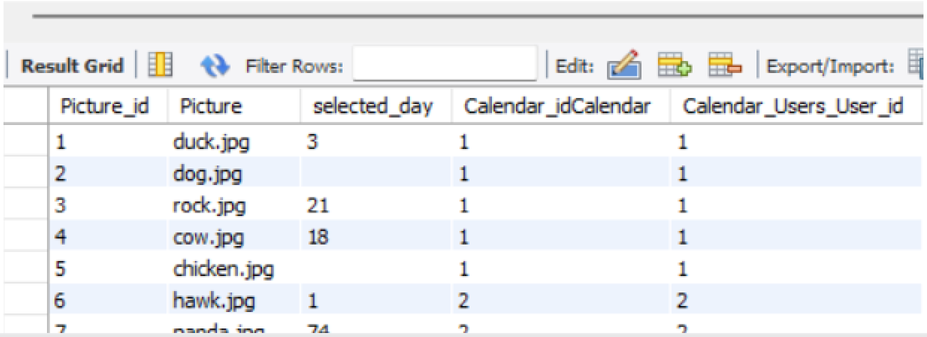
#	Time	Action	Message
5	01:14:49	INSERT INTO `calendardb`.`followers_list` (`Users_User_id`, `Users_User_id1`, `date_following`) VALUES ('10'...	1 row(s) affected

UPDATE

Update the picture in Picture_id = 5, which was “chicken.jpg”, to “eagle.jpg”

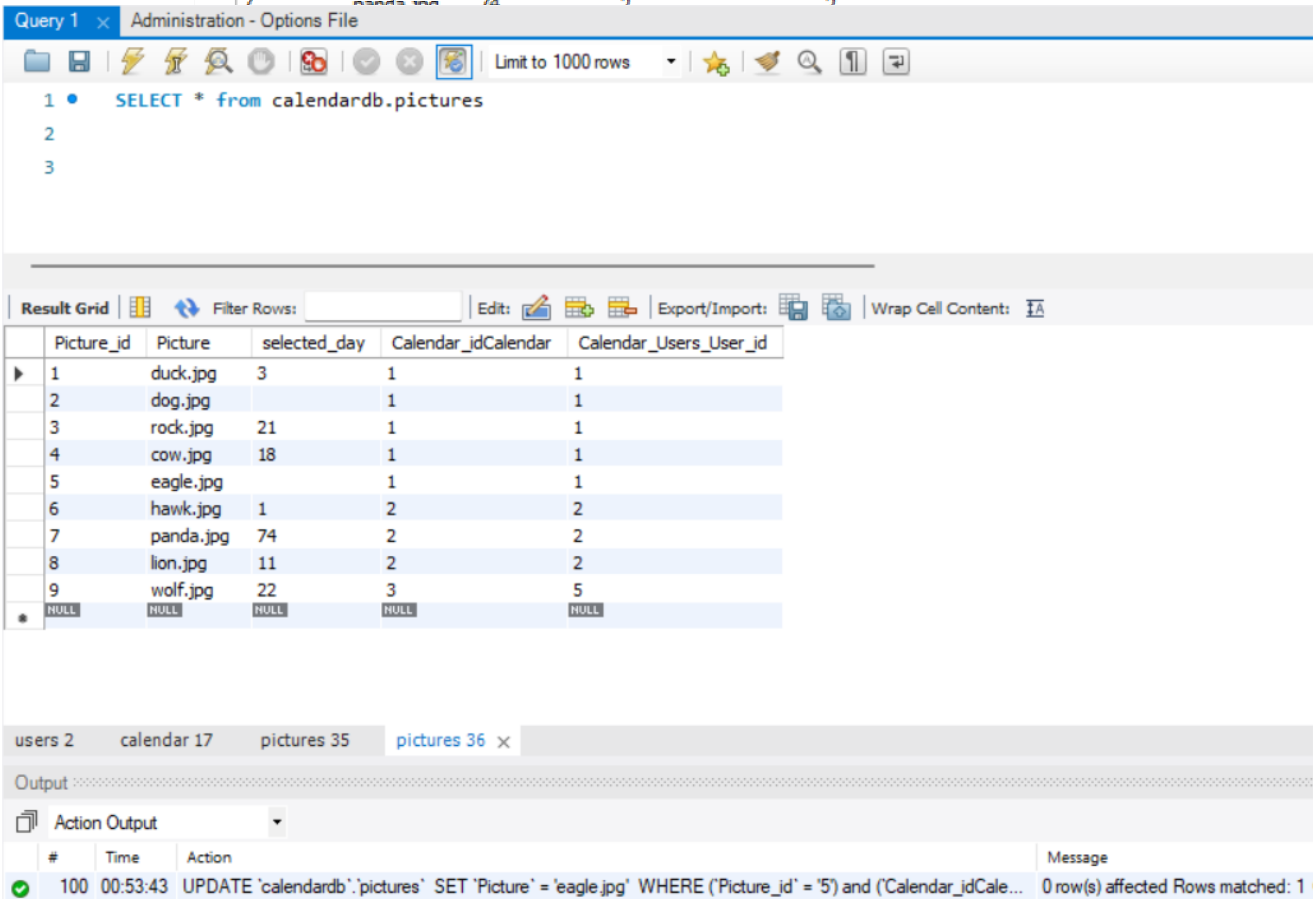


```
UPDATE `calendardb`.`pictures`
SET `Picture` = 'eagle.jpg'
WHERE (`Picture_id` = '5');
```



→

Picture_id	Picture	selected_day	Calendar_idCalendar	Calendar_Users_User_id
1	duck.jpg	3	1	1
2	dog.jpg		1	1
3	rock.jpg	21	1	1
4	cow.jpg	18	1	1
5	chicken.jpg		1	1
6	hawk.jpg	1	2	2
7	panda.jpg	74	2	2



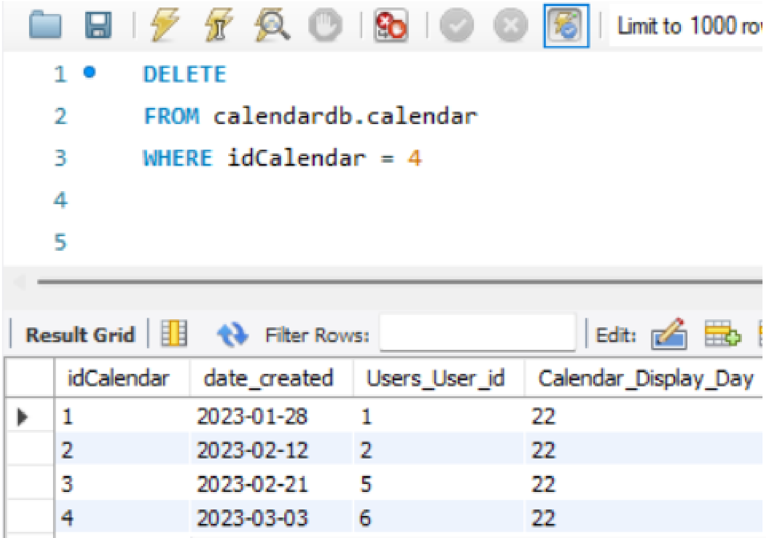
```
SELECT * from calendardb.pictures
```

Picture_id	Picture	selected_day	Calendar_idCalendar	Calendar_Users_User_id
1	duck.jpg	3	1	1
2	dog.jpg		1	1
3	rock.jpg	21	1	1
4	cow.jpg	18	1	1
5	eagle.jpg		1	1
6	hawk.jpg	1	2	2
7	panda.jpg	74	2	2
8	lion.jpg	11	2	2
9	wolf.jpg	22	3	5
NULL	NULL	NULL	NULL	NULL

#	Time	Action	Message
100	00:53:43	UPDATE `calendardb`.`pictures` SET `Picture` = 'eagle.jpg' WHERE (`Picture_id` = '5') and (`Calendar_idCale...	0 row(s) affected Rows matched: 1

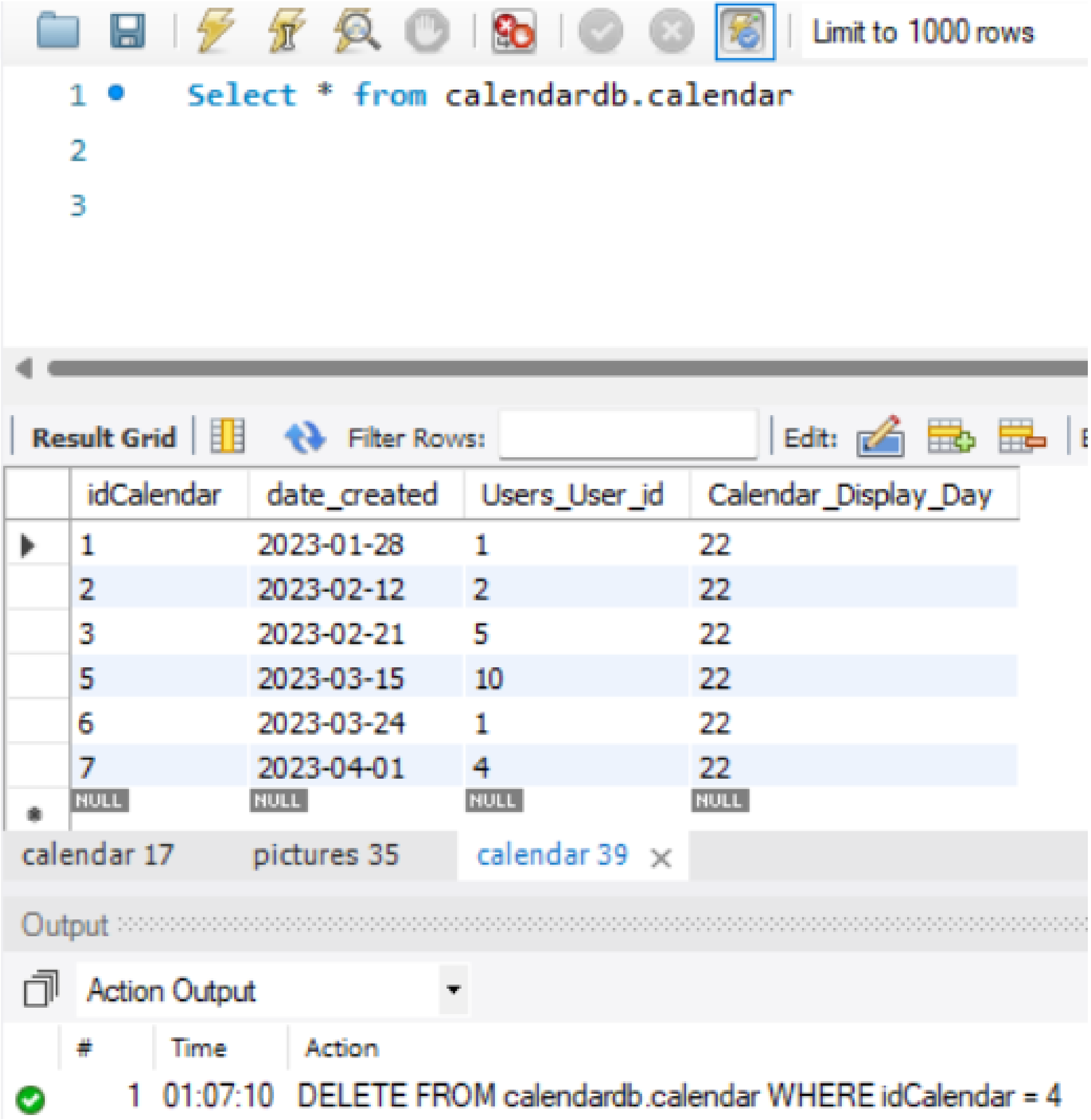
DELETE

Delete a calendar from the calendar table.



```
DELETE
FROM calendardb.calendar
WHERE idCalendar = 4
```

idCalendar	date_created	Users_User_id	Calendar_Display_Day
1	2023-01-28	1	22
2	2023-02-12	2	22
3	2023-02-21	5	22
4	2023-03-03	6	22
5			
6			
7			



```
Select * from calendardb.calendar
```

idCalendar	date_created	Users_User_id	Calendar_Display_Day
1	2023-01-28	1	22
2	2023-02-12	2	22
3	2023-02-21	5	22
5	2023-03-15	10	22
6	2023-03-24	1	22
7	2023-04-01	4	22
NULL	NULL	NULL	NULL

```
#   Time       Action
1   01:07:10   DELETE FROM calendardb.calendar WHERE idCalendar = 4
```

Conclusions

After completing this project, and this course, I have a new respect for SQL and databases. In the project alone, I learned the importance of functional diagrams/models before trying to turn them into actual databases. I found the MySQL WorkBench forward engineering very interesting. When I first learned SQL, we had to build, create, and seed all of our SQL tables through terminal commands. It was very interesting to see how easily it could take a good logical model and turn it into a working database quickly.

My second takeaway, and I was warned by the TA, is that a social media database is quite challenging, especially with the strictness of SQL's relationships. I found the challenge tricky at times, especially when trying to use a separate follower entity, but I enjoyed the process of thinking through each relationship. I felt like the part where we finally had to insert data into created tables was the most valuable part to this project. If I had a suggestion, it would be to have students give a diagram or draw on paper the exact tables and columns they expect to get out of their project. Visually seeing those helped me decide the best route to fix the follower entity and just make it another user entity that is connected to another user in a table.

This class definitely caught my interest in databases, and as I worked through this project, I researched popular social media applications and looked at what back-ends and database systems they were using. Twitter was originally built with SQL and the common post, comment, like build is still used to teach young coders today. The most impressive part to me, which we touched on and watched a video on during class, was the Hadoop storage. Really, any massive database storage management is impressive to me, such as SnapChat saving over 1.5 trillion pictures using AWS S3 cloud services.

Finally, I have tried to build this application before on a NoSQL database (MongoDB) and it was a cool experience to look at different pain points caused by both an SQL and a NoSQL database. I appreciate the effort that this project made me put forth into looking at databases with all sorts of perspectives, especially with better diagram building skills now.