INFS1200 Assignment 2 – Module 3

Code and PDF Due: Friday 10th October @ 3 PM AEST
Oral Assessment: Week 12, 20-24 October 2025
Weighting: 25%

Full Name:	**Student ID (8 digits):**
Fill in your full name here	*Fill in your student ID here*

Overview

This assignment tests your ability to use and apply SQL concepts to complete tasks in a real-world scenario. Specifically, this assessment will examine your ability to use *Data Query Language* (DQL) to retrieve data, *Data Manipulation Language* (DML) to create, update and delete data, and *Data Definition Language* (DDL) to create and update metadata. This assignment is to be completed **individually.**

Submission

Assignment 2 is made up of two parts.

Part 1

Part 1 will be submitted through an electronic marking tool called Gradescope. For this assignment, you will need to submit **two** types of files to the Gradescope portal:

Query Files

For each question in Sections A, B, C, as well as one question in section D, you are required to submit a separate *.sql* file which contains your SQL solution for that question. Empty SQL files will be available on Blackboard for you to fill in.

Each file should only contain the SQL query(s) and **no additional text or comments**.

Each file should be named as per the information in the question (e.g. *A1.sql*).

You may use any valid MySQL functions or syntax features, including any that weren't used in class.

Your queries will be autograded, so they **must compile in the autograder environment**, among other requirements; see below regarding the autograder.

There are no style marks for your SQL code, but please try to keep your queries readable.

Assignment PDF

You will need to make the following changes to this Word document:

- For Section A, include a screenshot of the output of your query for each question. This output should be produced from the released dataset (IntroInfoSystemsA2Release.sql) provided on Blackboard.
- For Section D, you must enter your answers in this document where required.

For Section A, include a screenshot of the output of your query for each question. This output should be produced from the released dataset (IntroInfoSystemsA2Release.sql) provided on Blackboard. This screenshot should show all data returned in a tabular format, as per the example below. Note that your output screenshot must match the SQL code that you submit. Marks are awarded based on the output generated by the autograder. A correct screenshot alone does not guarantee that your code produces the correct output or that marks will be awarded.

When your answers are filled in, export this document to a PDF and also **upload the PDF** to the Gradescope autograder portal. **Name this file *Assignment_2.pdf*.** Please do not alter the layout of this document. Changes that do not alter the layout of the document (e.g. highlighting) are acceptable. Ensure the name and SID boxes are completed.

Submission Instructions

1) Optionally, compress all of your SQL/txt files and your assignment PDF together. Compress the files directly; do not compress a folder that contains these files.
2) Submit to Gradescope (to the correct assignment portal):
    a. either all of your SQL/txt files and your assignment PDF together in a single submission; or
    b. your ZIP file.
3) **Check your autograder results** (see Autograder tests and feedback). You must check and understand the autograder results to pass this assignment.
4) If you have used generative AI for this assignment, also submit documentation of your AI usage to the other Gradescope portal (see Generative AI Usage).

You may resubmit as many times as you want to before the due date. Only your latest submission before the due date will be considered for final marking.

Part 2

Part 2 is an oral assessment that will be completed during a short, in-person interview with a demonstrator during your applied class in Week 12 (after your Gradescope submission). This interview is to verify your understanding of the code you submitted in Part 1 for sections A, B and C.

You will be required to explain the work you have submitted in Part 1 and discuss your choices.

All oral assessments must be given live and will be recorded by the teaching team (on Zoom) for archiving purposes. If you cannot attend your week 12 applied class, email the course coordinator immediately.

Ensure you have your **student ID card** as it will be required to verify your identity for this assessment. If you don’t, arrange for one immediately.

Marking

Assignment 2 is worth **25 course marks**. The marks available per section of Part 1 are as follows:

	Marks
Section A – SQL DQL (SELECT)	15 marks
Section B – SQL DML (UPDATE, INSERT, DELETE)	4 marks
Section C – SQL DDL	4 marks
Section D – Critical thinking	2 marks

Part 2 will receive a pass or fail result. You **must pass Part 2**, the oral assessment, to be eligible to pass Assignment 2. Failure in Part 2 will result in your assignment 2 mark being capped at 50% (12.5 marks). If you pass part 2, your final grade will be your grade from Part 1.

Sections A, B, C and a portion of section of D of this assignment will be graded via an autograder deployed on Gradescope. However, we reserve the right to revert to hand marking using the PDF submission should the need arise.

Late penalties

Please consult the course profile for late penalties that apply to this assessment item.

Autograder tests and feedback

The autograder will start grading your work automatically after each submission. This may take up to a few minutes to complete. As soon as possible after submitting and before the due date, you must check your autograder results. The autograder will run *visible tests* (results shown before the assignment due date) as well as *hidden tests* (results shown when assignment results are published). **It is your responsibility to ensure you understand how the autograder works and how to interpret the test results. If you do not, ask a course staff member as soon as possible**.

There are two forms of *visible tests*:

- *File existence and compilation tests*: Your SQL files must be named according to the information provided in each question, otherwise they won't be detected by the autograder and will receive 0 marks.
  Additionally, your code will be checked to see if it compiles correctly. Your queries must compile using **MySQL version 8.0 with default Linux settings**. This might not match the setup on your personal device, so it is highly recommended to test your queries using *phpMyAdmin* on your zones as shown in class. **Code that fails to compile in the autograder environment will receive 0 marks. No manual regrading will be made past the assessment deadline**. No marks are given for passing the compilation tests. **It is possible for queries to compile in your local environment but not in the autograder. The autograder results are what will contribute towards your final grade.**
- *Column domain checking*: The autograder will check whether the domains of the columns in your query results match the expected domains. Make sure to carefully read the *'Explanation*' section in each question of the assignment. This information will either state the expected domains of each column or allow you to infer them.

There are also *hidden tests* that will contribute towards your final grade, i.e. **passing all visible tests will not guarantee a grade of 100%**. These will test special cases that aren't covered by the visible tests, e.g. alternative datasets. All of these special cases will still conform with the given contextual information, relational schema, ER diagram and SQL DDL information. However, the instance data used for the edge cases will likely differ from the provided dataset.

Plagiarism

The University has strict policies regarding plagiarism. Penalties for engaging in unacceptable behaviour range from loss of grades in a course through to expulsion from UQ. You are required to read and understand the policies on academic integrity and plagiarism in the course profile (Section 6.1). If you have any questions regarding an acceptable level of collaboration with your peers, please see either the lecturer or your demonstrator for guidance.

Generative AI Usage

You are permitted to use generative AI tools to help you complete this assessment task. All interactions (both prompts and responses) must be submitted in the Generative AI submission portal provided on gradescope. An AI documentation template will be provided on Blackboard; see that document for instructions regarding how to reference Generative AI for this assignment. If you have not used generative AI, you do not need to submit anything in the Generative AI submission portal.

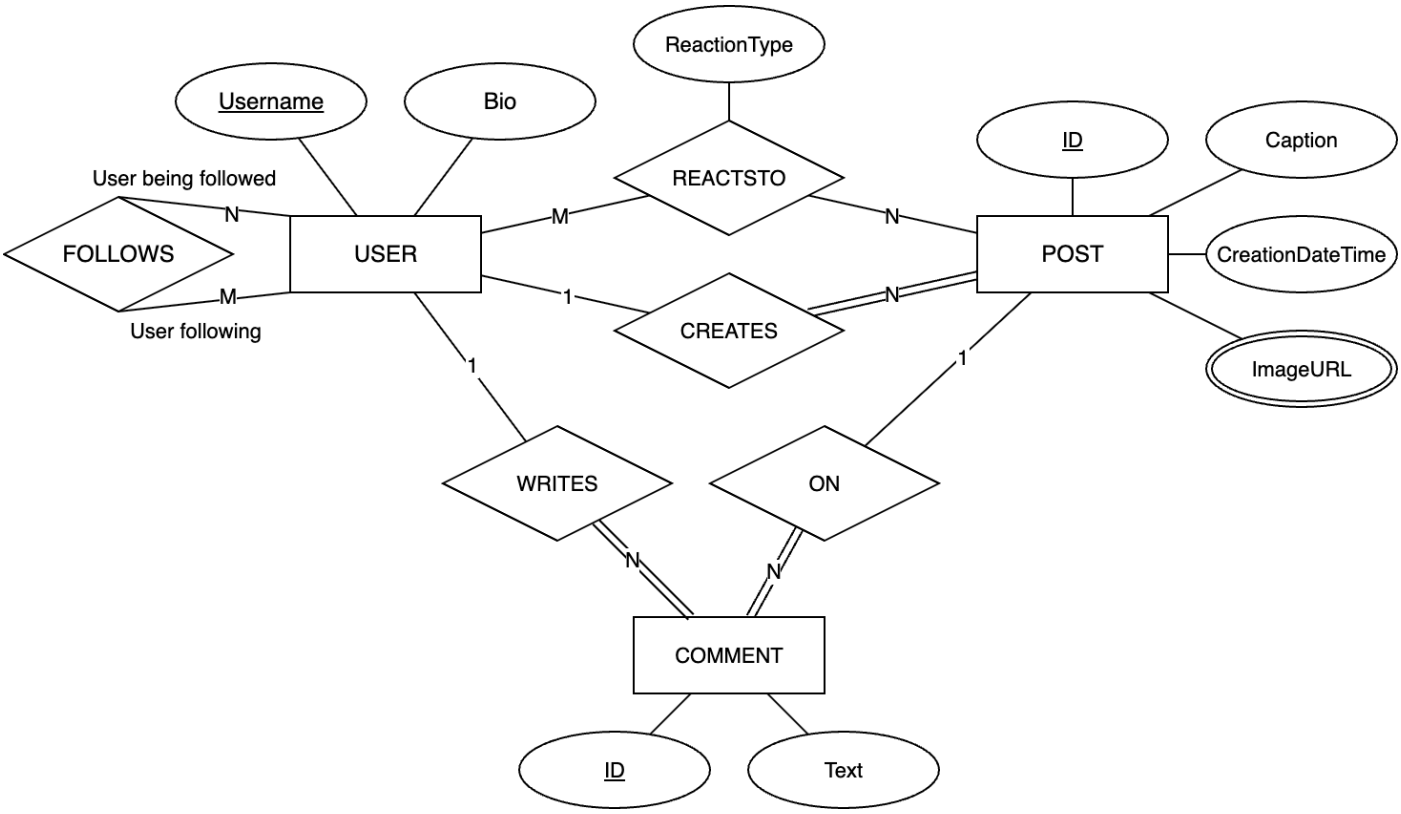
Please note that if you use generative AI but fail to acknowledge this by attaching your interaction to the end of the assignment, it will be considered misconduct, as you are claiming credit for work that is not your own.

Task

For this assignment, you will be presented with the schema of a social media platform's database.

AntiSocialMedia have designed and developed a simple relational database to manage their users and content. Users of the platform can create posts. Users may follow other users on the platform. Users can also react to posts or write comments on them.

ER Diagram



Relational Schema

User [Username, Bio]

Comment [Id, Text, Username, PostId]
Comment.Username references User.Username
Comment.PostId references Post.Id

Post [Id, Caption, CreationDateTime, Username]
Post.Username references User.Username

PostImage [PostId, ImageURL]
PostImage.PostId references Post.Id

UserFollows [UserBeingFollowed, UserFollowing]
(If user A follows user B, there will be an entry in this table where userBeingFollowed matches user B's username and userFollowing matches user A's username.)
UserFollows.UserBeingFollowed references User.Username
UserFollows.UserFollowing references User.Username

PostReaction [Username, PostId, ReactionType]
PostReaction.Username references User.Username
PostReaction.PostId references Post.Id

Example Question

Retrieve the username and bios of all users.

This query must return two columns: the usernames, and the bios of users.

Solution:

```
SELECT username, bio FROM User;
```

Output screenshot:

username	bio
Ahuman	Ahuman
Allan	A famous student at UQ who likes Operations Resear...
Daniel	A UQ tutor.
Ian	An computer science GOAT who teaches INFS1200 coou...
TotallyNotABot	Not A Bot.

Section A – SQL DQL (SELECT)

Ensure you have read the above pages to understand key information about the assignment and grading process. Understanding this information will be required to achieve a satisfactory grade for this assignment. If any part of this information is unclear, it is your responsibility to seek help from course staff as soon as possible.

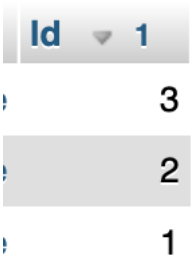
Question A1

Retrieve all of the comment IDs in descending order.

This query must return one column: the comment IDs.

Filename: A1.sql

Output screenshot:



Question A2

For all posts in the database, return the IDs and the number of comments.

This query must return two columns: the post IDs, and the number of comments for each post.

Filename: A2.sql

Output screenshot:

PostId	COUNT(*)
1	2
2	2

Question A3

Count the number of posts which have image(s) that have been uploaded onto *AntiSocialMedia's* server *https://antisocial.media*

Note: this means the url of the image must start with *https://antisocial.media*

This query must return one column: a single integer value.

Filename: A3.sql

Output screenshot:

num_posts
1

Question A4

Posts that have received heart reactions from at least 3 users are considered “popular”. For each “popular” post, find the post ID, and the number of users who have *heart* reacted to it.

This query must return two columns; the post’s ID, and the number of users who reacted to it.

Filename: A4.sql

Output screenshot:

PostId	COUNT(*)
1	3

Question A5

Which user(s) have made the most posts?

This query must return one column: the username of the user(s) who have made the most posts.

Filename: A5.sql

Output screenshot:

Question A6

Bot accounts are users who are not followed by any other users. Find the usernames of all suspected bot accounts.

This query must return one column: the usernames of bot accounts.

Filename: A6.sql

Output screenshot:

Username
TotallyNotABot

Question A7

Find all users who have commented on at least all the same posts that Ian has commented on. Ian can be identified in the database by the username *Ian*.

You may assume that Ian has made at least one comment. *The username Ian should be returned in this result.*

This query must return one column: the usernames of the users.

Filename: A7.sql

Output screenshot:

Username
Ian
Allan

Question A8

Allan wants to find all of his followers who did not react to his most recent post.

You may assume that no two posts can be created at the same time, and that Allan has created at least one post. Allan can be identified in the database by the username *Allan*.

You must use at least one **sub-query** to answer this question.

This query must return one column: the username(s) of users.

Filename: A8.sql

Output screenshot:

UserFollowing
TotallyNotABot

Question A9

Allan wants to find how many suspected “stalkers” he has on the *AntiSocialMedia* platform. A stalker would satisfy **all** of the following criteria:

1. Is following Allan;
2. Has reacted to at least one of Allan’s posts (you may assume that Allan has made at least one post); and
3. Has not commented on any of Allan’s posts (to avoid suspicion).

You must use at least one **set operation** and create at least one **view** in your answer.

This query must return one column: a single integer value for the number of stalkers Allan has.

Filename: A9.sql

Output screenshot:

NumStalkers
1

Section B – SQL DML (INSERT, UPDATE, DELETE)

Question B1

Allan made a post so popular that the database couldn't handle it, so you will need to insert it manually. This new post has an ID of 200 and a caption of "New pics from Japan!" (without the quotes). It was posted on the 10th of July this year at 2:53 PM. It has two images with URLs *https://antisocial.media/f1e2d3c4-b5a6-4978-9182-3d4e5f6a7b8c.png* and *https://antisocial.media/2c3d4e5f-6a7b-48c9-d1e2-f3a4b5c6d7e8.jpg*. Every user in the database, *other than Allan himself*, liked the post (i.e. a reaction type of "like"). The post has no comments.

You may assume that Allan exists in the database and has the username *Allan*.

Filename: B1.sql

Question B2

A malicious bot account with username *TotallyNotABot* has taken over an innocent account and caused inconsistencies in the database. All of the bot's reactions, posts and comments should be attributed to Daniel instead. All the records where *TotallyNotABot* has followed, or been followed by another account must be deleted from the database.

Perform all these changes and then finally remove the bot account from the database.

You may assume that accounts with usernames *TotallyNotABot* and *Daniel* exist. You may assume there are no posts that both Daniel and TotallyNotABot have reacted to.

Filename: B2.sql

Section C – SQL DDL

Note that since these questions will be autograded, the table and attribute names in your submitted code must exactly match the given information, including correct capitalisation.

Question C1

The platform wants to add an instant messaging system. Start by adding a *Message* table to the database. The Message table has the following attributes:

- *Id*: an integer, which uniquely identifies messages.
- *Sender*: a reference to the sending user's username.
- *Receiver*: a reference to the receiving user's username.
- *Message:* the message sent. The message cannot be longer than 512 characters.
- *IntegrityHash*: A SHA-256 hash of all details relating to the message, which is generated and stored with the message. The hash is exactly 64 characters long and must be unique for each message.

Each message must have a sender and receiver. Users cannot send messages to themselves.

Filename: C1.sql

Question C2

Add an attribute called *dob* to the *User* table to store a user's date of birth. A user's date of birth cannot be NULL. For existing users, set their date of birth to 1st January 1970.

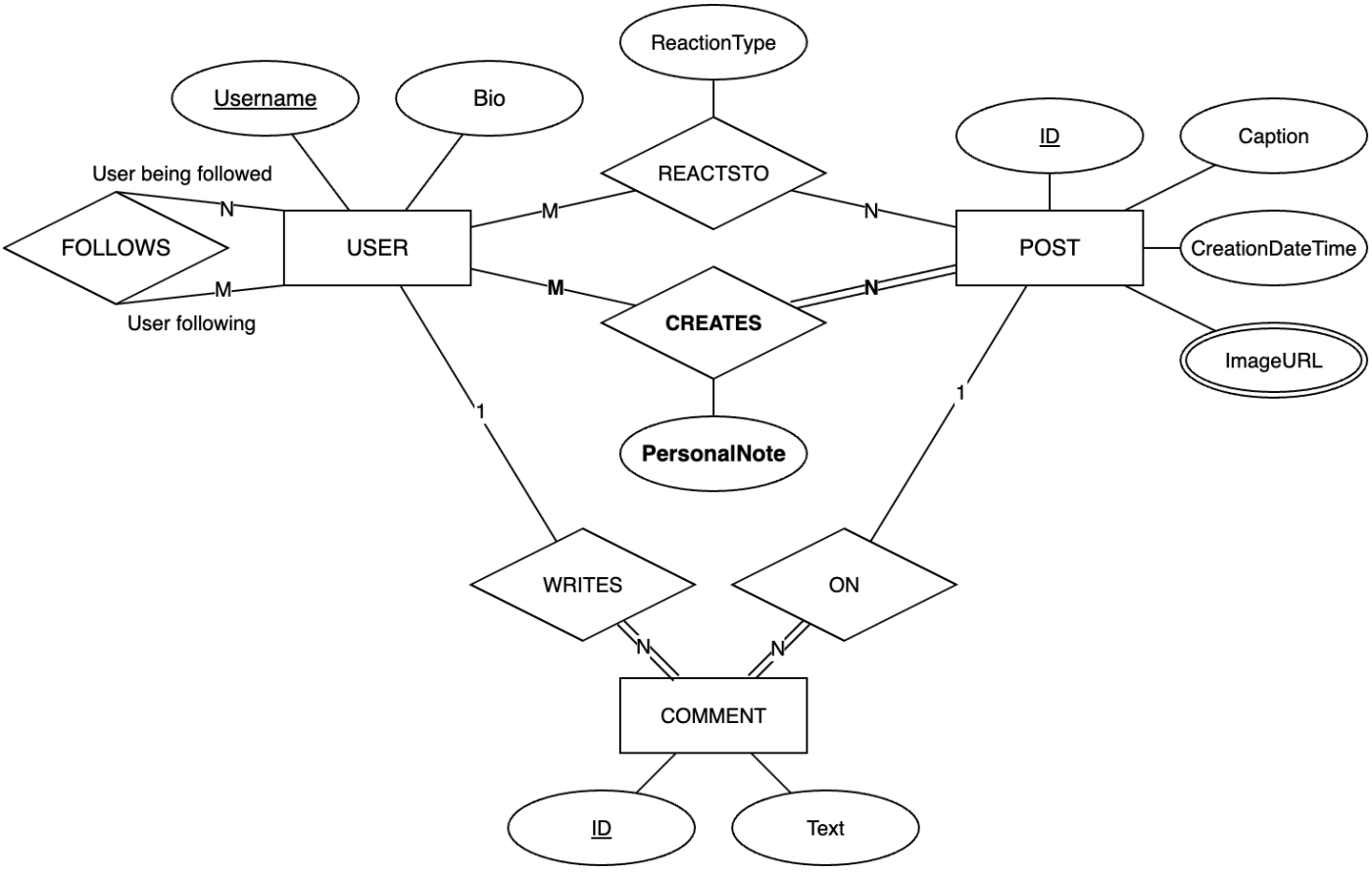
Filename: C2.sql

Section D – Critical Thinking

In this section, you will receive theoretical situations related to the UoD mentioned in the task description. Your task is to offer strategies to tackle the situation and write SQL queries to execute the approaches, where applicable.

Question D1

The database is being changed to allow co-authorship of posts, i.e. a single post can now be created by multiple users. Each post author can also add their own personal note to the post. The updated ER diagram is as follows:



The following new relation will need to be created:

UserCreatesPost [Username, PostId, PersonalNote]
UserCreatesPost.Username references User.Username
UserCreatesPost.PostId references Post.Id

For ***existing*** posts (note that existing posts would only have one author), their previous author must be maintained by creating a new personal note of the format “AUTHOR: CAPTION” where AUTHOR is the author’s username and CAPTION is the original caption of the post. For example, if Ian had a post captioned “Felt cute, might delete later” then Ian’s personal note is “Ian: Felt cute, might delete later”. You do not need to worry about two or more users posting together and may assume they will automatically write their own structured PersonalNote.

Your task is to update the database, including any applicable constraints, and transfer existing data as required. **Submit a separate SQL file for this question and name it D1.sql**.

This question will be autograded, so the table and attribute names in your submitted code must exactly match the given information, including correct capitalisation.

You do *not* need to enforce the total participation constraint on the POST side of the CREATES relationship.

Filename: D1.sql

Question D2

AntiSocialMedia wants to provide friend recommendations to all users. Specifically, each user should get up to five user recommendations. This list should be in ascending order from the first to fifth best friend recommendations. If there are less than five users in the database, rank all of them.

With the given contextual information about the platform in mind, describe a strategy regarding how you would rank them from 1 to 5.

Write an SQL query to retrieve the usernames of recommended friends (in order) for the user with username “Allan” (without the quotes).

Do *not* submit a separate file for this query. This query will *not* be autograded.

Your explanation must be at most **200 words** long (excluding SQL code).

Strategy:

Friend recommendations should rely on UserFollows information primarily. Firstly, Allan should never be recommended as a friend. Even “if there are less than five users in the database”, I will not return Allan. The first ordering criterion is if Allan follows a user. If that is a tiebreak, the more mutuals the higher ranked. Finally, to boost engagement, the more followers a user has the higher they will be ranked. To avoid non-deterministic orderings, the remaining tiebreaks will be split by alphabetically ordering people’s usernames. Since usernames are used as the primary key of Users, this is guaranteed to break all ties and provide a well-ordering of all users.

The following SQL seems reasonable to me, and I have included it in D2.sql in addition to below against your express intention. Note that true > false in mySQL 8.4 for ordering purposes.

SQL solution:

```
-- Number of followers by username
CREATE OR REPLACE VIEW NumFollowers AS
	SELECT User.Username, COUNT(UserFollows.UserBeingFollowed) as FollowerCount FROM
User
		LEFT JOIN UserFollows ON UserBeingFollowed = User.Username
		GROUP BY User.Username;

SELECT * FROM NumFollowers;

-- For each Username, how many people they follow that Allan also follows
CREATE OR REPLACE VIEW MutualsWithAllan AS
	SELECT User.Username, Count(UserFollows.UserFollowing) as MutualsCount FROM User
		LEFT JOIN UserFollows ON UserFollows.UserFollowing = User.Username
			AND UserFollows.UserBeingFollowed IN
				-- All people who Allan follows
				(SELECT UserBeingFollowed FROM UserFollows WHERE UserFollowing =
"Allan")
		GROUP BY User.Username;

SELECT * FROM MutualsWithAllan;

-- The people who follow Allan
SELECT UserFollowing FROM UserFollows WHERE UserBeingFollowed = "Allan";
-- The people who Allan follows
SELECT UserBeingFollowed FROM UserFollows WHERE UserFollowing = "Allan";

-- Rank every user except Allan himself
SELECT User.Username, User.Bio,
	User.Username IN
		(SELECT UserFollowing FROM UserFollows WHERE UserBeingFollowed = "Allan")
	as AllanFollows,
MutualsCount, FollowerCount FROM User
	LEFT JOIN MutualsWithAllan ON MutualsWithAllan.Username = User.Username
	LEFT JOIN NumFollowers ON NumFollowers.Username = User.Username
	WHERE User.Username != "Allan" -- Don't ever suggest Allan himself
	ORDER BY
		User.Username IN -- prioritise if user follows Allan
		(SELECT UserFollowing FROM UserFollows WHERE UserBeingFollowed = "Allan") DESC,
		MutualsWithAllan.MutualsCount DESC, -- Mutuals first
		NumFollowers.FollowerCount DESC, -- tiebreaker who follows the most people
		User.Username
	LIMIT 5;
```