Team Members: Zoe Robinson (zrobinso), Alex Torres Vivaldo (atorresv)
Team T1-16

Discord Phase 2

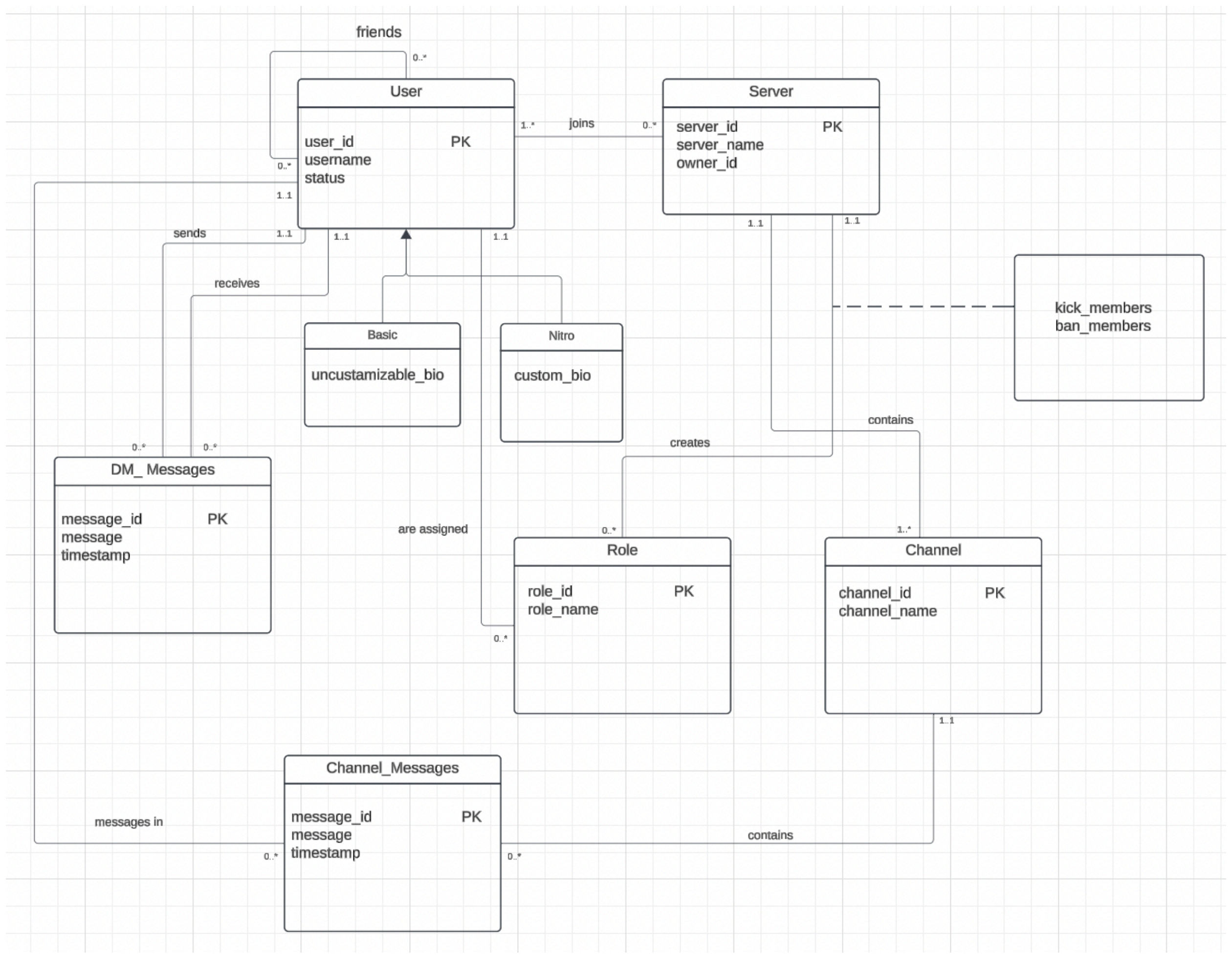
Discord Three Users:
1. Server Owner
2. Server Member
3. Discord Manager

Updated User Stories:

ID	Verb+Noun	As a <role>	I want <goal>	So that <reason>	Simple / Complex	Operational / Analytical
US0	Set status	Server Member	To set my status to Do Not Disturb	My friends know not to bother me	Simple	Operational
US1	Join a server	Server Member	Join a server	I can engage in a community	Simple	Operational
US2	Get role names	Server Owner	Get a list of role names within my server	I can see if any are out of date	Complex	Operational
US3 (initialized for phase 2)	Get Usernames	Server Owner	Filter members that have a specific role	Look for specific members efficiently	Complex	Operational
US4	Find Friend	Server Member	To see which of my friends are online	I can easily message them without bothering them	Complex	Operational
US5 **(new)**	See User Statistics	Server member	view my activity	I can manage	Complex	Analytical

			statistics: total num messages sent + # servers involved in.	my time accordingly.		
US6	Find percent Nitro	Discord Manager	Find the percentage of Discord users that have nitro	I can edit marketing strategy accordingly	Complex	Analytical
US7 **(Trigger Function)**	Kick user	Server owner	To remove roles from server members when I kick them.	I can uphold the guidelines of the server	simple	Operational
US8	Add friend	Server Member	To add my friend on discord	I can message and call them on discord	Simple	Operational
US9 **(Window Function)**	Calculate cumulative messages sum + per user	Discord Manager	To produce a cumulative sum of messages for all users in specified channel	So that I can track activity of channel, along with discord members	Complex	Analytical

Conceptual Model:



Converting to Relational Model

Servers(**server_id**, server_name, owner_id)
Discord_Users(**user_id**, username, status)
DM_Messages(**message_id**, message, timestamp, <u>user_id1, user_id2</u>)
Basic(<u>**user_id**</u>, uncustomizable_bio)
Nitro(<u>**user_id**</u>, custom_bio)
Channel(**channel_id**, channel_name, <u>server_id</u>)
Channel_Messages(**message_id**, message, timestamp, <u>channel_id</u>, <u>sent_by</u>)
Server_Members(<u>**server_id**</u>**,** <u>**user_id**</u>) ← Created from the joins many to many relationships
Role(**role_id**, role_name, kick_members, ban_members, <u>server_id</u>, <u>user_id</u>)
Friends(<u>**user_id1**</u>, <u>**user_id2**</u>)

Functional Dependencies

Servers(server_id, server_name, owner_id)

1. Server_id → server_name, owner_id
    - Server_id+ = {server_id, server_name, owner_id} → Good FD

This functional dependency is valid because a server has only one owner and one name, thus we can determine those attributes just by knowing the server_id. This functional dependency is also good because it is a super key that is able to access all columns, meaning server_id is a primary key. Therefore, this relation is already in BCNF (No partial/transitive dependencies).

Discord_Users(user_id, username, status)

1. user_id → username, status
    - User_id+ = {user_id, username, status} → Good FD

This is a valid functional dependency because every user has their own username and their status can be directly derived from user_id. This functional dependency is also good because the LHS is a super key that can access all other fields within the relation, meaning user_id is a primary key. It is also a candidate key because it is minimal. Thus, this relation is already in BCNF (No partial/transitive dependencies).

Server_Members(user_id, server_id)

There are no functional dependencies that can be derived from the fields within the entity. When decomposing the relation to BCNF, we find that the super key is instead just the composite key containing user_id and server_id: Server_Members(**user_id**, **server_id**). Therefore, this relation is in BCNF.

Role(role_id, role_name, kick_members, ban_members, server_id, user_id)

1. role_id → role_name, kick_members, ban_members
    - role_Id+ = {role_id, role_name, kick_members, ban_members} → Bad FD

(1) is a bad functional dependency because the LHS is not a super key since not all attributes can be accessed through role_id.

This relation is also currently in 1NF because role_id is a partial dependency as it is part of the following candidate key: server_id, user_id, role_id.

Basic(user_id, uncustomizable_bio)

1. user_id → uncustomizable_bio
    - user_id+ = {user_id, uncustomizable_bio} → Good FD

This is a valid functional dependency in our context because if we know the users that have the basic subscription to discord, we will know their uncustomizable bio. Additionally, this functional dependency is good because the LHS is a super key. Since the relation is already in BCNF form(No partial or transitive dependencies), that means that user_id is a primary key in this relation.

Nitro(user_id, custom_bio)

1. user_id → custom_bio
    - user_id+ = {user_id, custom_bio} → Good FD

Similarly, if we have the users who are subscribed to the Nitro membership on discord, we are able to identify their custom bio. This functional dependency is good because the LHS is a super key. Since the relation is already in BCNF, that means the user_id is the primary key.

Channel(channel_id, channel_name, server_id)

1. channel_id → channel_name, server_id
    - channel_id+ = {channel_id, channel_name, server_id} → Good FD

This is a good functional dependency because the LHS(channel_id) determines all other fields within the relation such as the name and the server it's in. Thus, this functional dependency is already in BCNF, meaning the primary key is still channel_id.

Channel_Messages(message_id, message, timestamp, sent_by, channel_id)

1. message_id → message, timestamp, sent_by
    - message_id+ = {message_id, message, timestamp, sent_by} → Bad FD

We derived this functional dependency because although message_id determines the message, timestamp, and the sender, we are not able to derive what channel it came from. Therefore, message_id is not a super key, making this functional dependency bad.

After decomposing this relation to BCNF form in the next page, we find that there exists as partial dependency in message_id, within the original relation, since the primary key for the new decomposed relation (also called Channel_Messages), is message_id and channel_id. Therefore the original relation above was in 1NF.

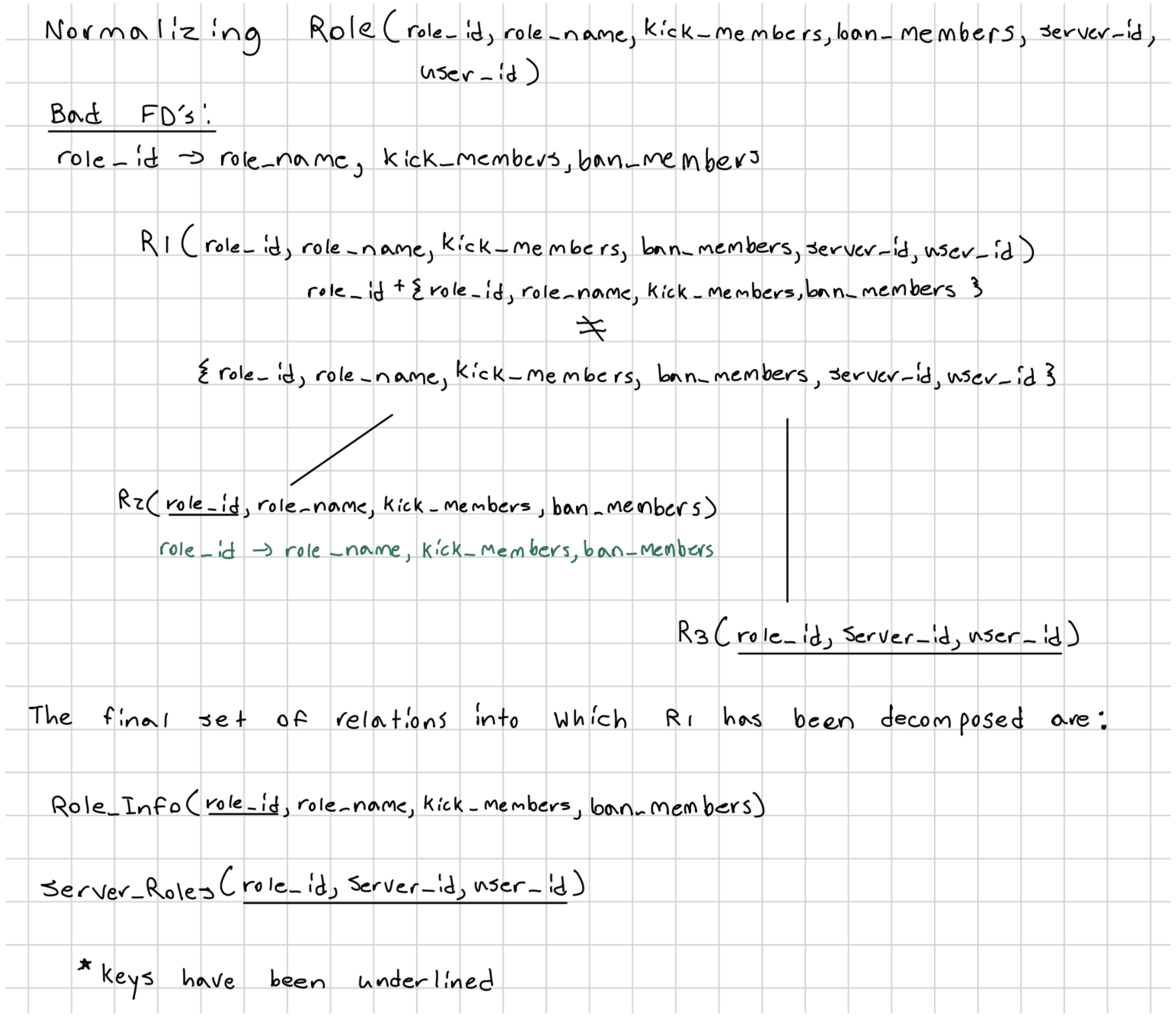
DM_Messages(dmessage_id, message, timestamp, sent_to, sent_by)

1. dmessage_id → message, timestamp, sent_to, sent_by
    - dmessage_id+ = {dmessage_id, message, timestamp, sent_to, sent_by} → Good FD

This is a valid functional dependency because through the key dmessage_id, we are able to determine the rest of the fields. Therefore, dmessage_id is a super key since all fields in the relation can be identified. Thus this relation is already in BCNF.

Normalization

Normalizing Role



Normalizing Role(role_id, role_name, kick_members, ban_members, server_id, user_id)

Bad FD's:

role_id → role_name, kick_members, ban_members

R1(role_id, role_name, kick_members, ban_members, server_id, user_id)

role_id$^+$ {role_id, role_name, kick_members, ban_members}

≠

{role_id, role_name, kick_members, ban_members, server_id, user_id}

R2(role_id, role_name, kick_members, ban_members)

role_id → role_name, kick_members, ban_members

R3(role_id, server_id, user_id)

The final set of relations into which R1 has been decomposed are:

Role_Info(role_id, role_name, kick_members, ban_members)

Server_Roles(role_id, server_id, user_id)

* keys have been underlined

Normalizing Channel Messages

R1(message_id, message, timestamp, sent_by, channel_id)
Bad FD:

message_id^+ {message_id, message, timestamp, sent_by

R1(message_id, message, timestamp, sent_by, channel_id)
message_id^+ {message_id, Message, timestamp, sent_by}

$\neq$

{message_id, message, timestamp, sent_by, channel_id}

R2($\underline{\text{message\_id}}$, Message, timestamp, sent_by)
message_id $\rightarrow$ message, timestamp, sent_by

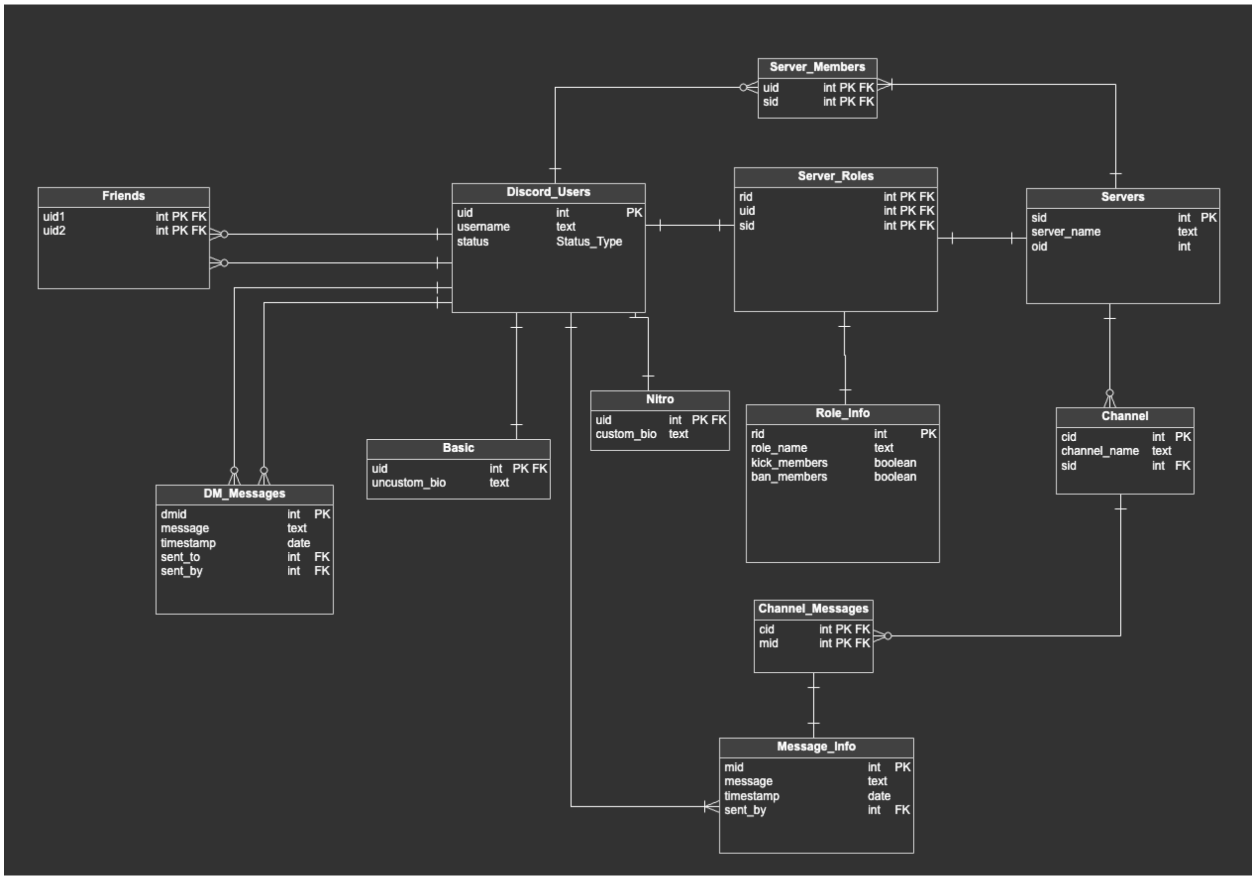
R3($\underline{\text{message\_id}}$, $\underline{\text{Channel\_id}}$)

The final set of relations into which R1 has been decomposed are:

channel_messages($\underline{\text{message\_id}}$, $\underline{\text{channel\_id}}$)
message_info($\underline{\text{message\_id}}$, message, timestamp, sent_by)

*keys have been underlined

Physical Model



Vertabelo Link: https://my.vertabelo.com/doc/JTzMkpT61Luuk7ofSBMUwSuQH8dtZFWA