Minute 1

In this video, we’ll learn how to design a
database for a chat application, such as WhatsApp.
Whether you’re watching this
for a university project,
or want to see an example of a database design,
or want to improve your skills, seeing a database
being designed to meet some requirements is a good
way to understand how the process can be done.
We’ll briefly cover what the chat application
is and what it needs to do, and create our
database design from scratch, enhancing
it with each requirement that we add.
Let’s get into it!
So, what is a chat application?
It’s an application, either
on the mobile or a computer,
that allows people to send
short messages to other people.
I mentioned WhatsApp before, and that’s the
kind of example we’ll use in this video.
Other examples are the Messages app on mobile
phones, or Facebook Messenger, or Signal.
Let’s say we’re designing the database for a
new application. What do we need to design?
We would start with the requirements,
which define what we want to do
or what we need to do. I’ll use a few sentences to
describe what we want this chat application to do.

Minute 2

You can use the same thing, or you can
come up with your own requirements.
Either way, we use them as a
starting point for our database.
So what are our requirements
for this chat application?
We want users to be able to use the application
to send messages to other people. Messages are
sent to a phone number which can be a number they
enter, or a contact from their contact list. The
contact list can be used to add people and phone
numbers that the user wants to send messages to.
For each contact, the user should be able
to specify their first and last name,
a profile photo image, and a phone number.
When a user sends a message, they can send a
text message only. We could look into audio,
video, GIFs, and other images later,
but for now we can just keep it simple.
The application allows for multiple
people to be in a group message,
and all people will be able to send messages to
the group and view messages within the group.
Groups can have names. If a
person is removed from a group,
they cannot see any messages sent after they
were removed. And if they are added to a

Minute 3

group after it is created, they cannot see any
messages before they were added to the group.
I think that’s enough for now. We can design
our database based on these requirements.
Let’s get started!
We can start with a table for messages.
We know that messages need to be stored in
our application, so let’s start with that.
We’ll create a table called “message”. This will
store the messages that we send to other people.
This table will have a field called
message_id which is the primary key.
You can call it something else if you prefer,
but as long as it’s a primary key, it’s OK.
Next, we’ll store the number that the
message is sent from. This will be your
phone number if you send the message, or
another person if they send a message.
We also have a to_number field which is
the number that a message is sent to.
The next field is the actual message that
is sent, which is called message_text.
Finally, we have the date and time the message
was sent, which is the sent_datetime field.

Minute 4

This is our simple message table. It allows us
to store information about the messages, and the
chat application can handle the functionality of
sending the messages and choosing the recipient.
What’s next?
In the requirements, we want the
ability to send messages to a contact.
So instead of specifying a phone number, you
can choose a contact in your address book.
Let’s add the ability to
store that in our database.
We’ve got a table here called contact.
This has a primary key called contact_id.
In our requirements we wanted
to store their first name,
last name, a profile photo,
and their phone number.
These fields were optional and the application
will allow the user to provide this information.
Now, how do we relate a message to a contact?
The requirements say that a message can be sent
to a number or a contact from their list. So,
a message does not need to relate to a contact.
How do we relate these two tables together?
There are a couple of ways we can do this.

Minute 5

If we add a contact_id to the
message table as a foreign key,
we can store messages that are sent to contacts.
This would mean the to_number can still be set,
which could be different from
the phone number for a contact.
This is OK though, as it could be a valid
situation. A contact is an up-to-date record
of a person, while a message is a point-in-time
record of something that happened. A contact could
change their phone number, but the phone number
for the original message would be unchanged.
Another way would be to not have the to_number in
the message table and look it up
from the contact table. However,
this would not allow for the database to store
the number a message was sent to at the time.
This may be a valid scenario. We
don’t know if that is important,
and if this was a real system,
we would want to check that.
For this video, we’ll assume we want to store the
number, so we’ll leave it in the message table.
So, we have a contact linked to
a message table. What’s next?
The next feature to add is group
messages. The requirements for this were:

Minute 6

multiple people can be in a group message,
all people can send and receive and view
messages in the group,
groups can have a name,
and some rules around members only seeing
messages while they are in the group.
So how do we add this to our database?
We can start with a table for the group.
We'll call the table message_group for now,
as the word group is a reserved word in SQL.
The table will have a primary key called
group_id, and we can add a name for the group.
Now we have our message_group table,
let's see what we can add next.
We have groups. We need to store the
members of each group. How can we do that?
A group can have multiple
people, or contacts inside it.
Also, a contact can be in multiple groups. For
example you may have one group with just your
siblings, so your brothers and sisters, and
you may have another group for your entire
family which would include brothers,
sisters, parents, children, and so on.
So, this is a many-to-many relationship. I created
a video on how to model these in a database,

Minute 7

which you can view in the description below,
and it’s done by creating a joining table
between the two tables we want to relate.
We would have a new table that contains a
record for a message_group and a contact.
We’ve called this group_member.
We can add in a foreign key for the
contact_id column in the contact table,
and the group_id column in
the message_group table.
We also need to add columns to show
when the person joined the group
and when the person left the group, so
the application knows which messages the
contact can see from the group. We’ve got the
joined_datetime and left_datetime for that.
So that’s how a contact can be related to a group.
Once we have the message_group created, we
need to be able to send messages to the group.
To do this, we can relate a message to the
message group. Add a foreign key to the
message table to join to the message_group
table, which can be called group_id.
We can see that our tables are now related. But
they are related in a circular way. Message,

Minute 8

to contact, to group member,
to message group, to message.
Sometimes this can be valid. But most of the
time it can indicate an issue in our database.
What could the issue be? We can look at the
message table. A message is related to a group.
It also has a contact. What if a message
is sent to a group and not a single person?
One way we can address this is by thinking
about the concept of a message group.
This table allows the concept of sending
a message to a group of contacts.
However, if we design this to
be more of a “conversation”,
where a message can be sent to one person or
multiple people, then we could make it work.
To do this, we can rename our table from
message_group to conversation. We would
also rename the columns in this table and in the
related tables to conversation instead of group.
We still have the circular
relationships between our tables.
Because the message is related to a conversation,
we don’t need to relate it to a contact.

Minute 9

So, we can remove the link
between contact and message,
which means removing the
contact_id from the message table.
We should also remove the to_number field because
we don’t send a message to a single number.
We can store all numbers that
we send messages to as contacts,
and the rest of the details in
the contact table can be optional.
The application can make this easy to do
for the user. The user can still enter a
single phone number for a message if they want,
but in the background a contact will be made.
This also brings us to our final
design for our requirements.
We’ve got the ability to message one
or more people, we can store contacts,
and the application can show messages to people
who are in a conversation at a point in time.
What issues might we have with this? It doesn’t
store the phone number a message was sent to,
because it may change. But the message is stored
against the contact, so it may not be an issue.
We could change the from_number to refer to a
contact instead of a phone number. This could

Minute 10

mean each user is a contact in the application.
But it’s not something we did for our design.
We could enhance the message_text
field to allow for other media types.
We could do this by changing the data type
to allow for files or references to files.
There are a few ways we could do that.
So those are some examples of how
we could enhance this database.
But the design we came up with will meet
our requirements for a chat application.
And that brings us to the end of this video on
designing a database for a chat application.
If you learned something new from this
video, make sure to subscribe to my channel.
If you want to learn more about database
design and development, visit databasestar.com.
That’s where I share my best
database-related content.
Which tip from this video was the
most helpful? Was it the fact that
we can make changes to things we
have already designed as we go,
or how we handled the many-to-many
relationship, or something else?
Thanks for watching.