

# GraphQL

## 1. Favourite Website: YouTube

## 2. Entitäten und Attribute:

- **Video:** id, title, description, like, dislike, uploaded\_at, user\_id, kanal\_id, url
- **Kanal:** id, name, description, created\_at, User\_id
- **Kommentar:** id, content, created\_at, user\_id, video\_id
- **User:** id, username, password, email, created\_at
- **Like:** id, user\_id, video\_id

## 3. Aktionen:

- Create a channel: Benötigte Entität: Channel, User
- Create a Video: Benötigte Entität: Video, Channel, User
- Video liken/disliken: Benötigte Entität: Video, Like, User
- Register a User: Benötigte Entitäten: User
- Fetch Videos: Benötigte Entitäten: Video, Channel, User
- Subscribe to a channel: Benötigte Entitäten: Channel, User

## 4., 5., 6. Aufgabe:

**Top Screenshot: Kanal Table Configuration**

Columns:

- id - integer, primary key, unique
- name - text
- description - text
- subscribers - integer
- created\_at - date
- user\_id - integer

Table Properties:

- Primary Key: id - Kanal\_pkey
- Foreign Keys: user\_id → User.id - Kanal\_user\_id\_fkey

*Links kann man die Tabellen sehen und rechts sind die Attribute angezeigt. Es wurden entsprechend für alle Tabellen Attribute hinzugefügt und die Beziehungen zwischen Tabellen sind geknüpft.*

**Bottom Screenshot: User Table Relationships**

NAME	SOURCE	TYPE	RELATIONSHIP	
Kanals	default	Array	public.User / id ↔ public.Kanal / user_id	<a href="#">Rename</a> <a href="#">Remove</a>
Kommentars	default	Array	public.User / id ↔ public.Kommentar / user_id	<a href="#">Rename</a> <a href="#">Remove</a>
Likes	default	Array	public.User / id ↔ public.Like / user_id	<a href="#">Rename</a> <a href="#">Remove</a>
Videos	default	Array	public.User / id ↔ public.Video / user_id	<a href="#">Rename</a> <a href="#">Remove</a>

*Hier sieht man beispielsweise die Beziehungen von User mit anderen Entitäten.*

## 7. Aufgabe:

Mutation -> Änderung der Daten auf dem Server (Schreibeoperation)

Query -> Datenabfrage vom Server (Leseoperation)

**Action- Create a Channel:** mutation CreateChannel(\$name: String!, \$description: String!, \$user\_id: Int!) { insert\_channel(objects: {name: \$name, description: \$description, user\_id: \$user\_id, subscribers: 0}) {  
 returning {  
 id  
 name  
description  
created\_at  
 subscribers }}}}

### **Action- Create a Video:**

mutation CreateVideo(\$title: String!, \$description: String!, \$url: String!, \$user\_id: Int!, \$channel\_id: Int!) {  
 insert\_video(objects: {title: \$title, description: \$description, url: \$url, user\_id: \$user\_id, channel\_id: \$channel\_id, likes: 0, dislikes: 0}) {  
 returning {  
 id  
title  
 description  
 url  
 uploaded\_at  
 likes  
 dislikes }}}}

### **Erklärung zum obigen Code:**

Mutation CreateChannel bzw. CreateVideo ist der Name der Mutation  
\$titel, \$name,\$description etc. repräsentieren die Parameter, während titel, name, description etc. die Datenfelder repräsentieren. Das Ausrufezeichen ! kennzeichnet diese Parameter als Pflichtfeld. Auch ist der Typ der Parameter vorgegeben. Insert\_video bzw. Insert\_channel ist der Name der Operation. Hier ist es das Einfügen von Datensätzen in die channel- bzw. Video-Tabelle. „objects“ ist ein Objekt, welches die neuen Datensätze beinhaltet und entsprechend setzt. Das „returning“ gibt die Datenfelder innerhalb der Klammer zurück. So kann man sehen, ob die Mutation am Server erfolgreich war oder nicht.

**Action- Like a Video:** mutation LikeVideo(\$user\_id: Int!,  
\$video\_id: Int!) { insert\_like(objects: {user\_id: \$user\_id,  
video\_id: \$video\_id}) { returning {  
id }}

**Action- Register a User:** mutation RegisterUser(\$username: String!, \$email: String!,  
\$password: String!) { insert\_user(objects: {username: \$username, email: \$email,  
password: \$password}) { returning {  
id  
username  
email  
created\_at }}}}

**Action- Fetch Videos from Max Mustermann:** query  
FetchVideosByUser {  
video(where: {user: {username: {\_eq: "Max Mustermann"}}}) {  
id  
title  
description  
url  
uploaded\_at  
likes dislikes user {  
username  
}  
channel { name  
subscribers }}}}

Query-Name ist FetchVideosByUser mit der Bedingung, dass der User Max Mustermann als username hat. Von ihm werden die Felder id, title, description, url, uploaded\_at, likes, dislikes abgefragt. Zusätzlich wird von ihm durch den Fremdschlüssel auf die Tabelle user sein username und von der Tabelle sein channel-name und subscribers abgefragt.

**Action- Subscribe to a channel:** mutation  
SubscribeChannel(\$channel\_id: Int!) {  
update\_channel(where: {id: {\_eq: \$channel\_id}}, \_inc: {subscribers: 1}) {  
returning {  
id  
subscribers  
}}}