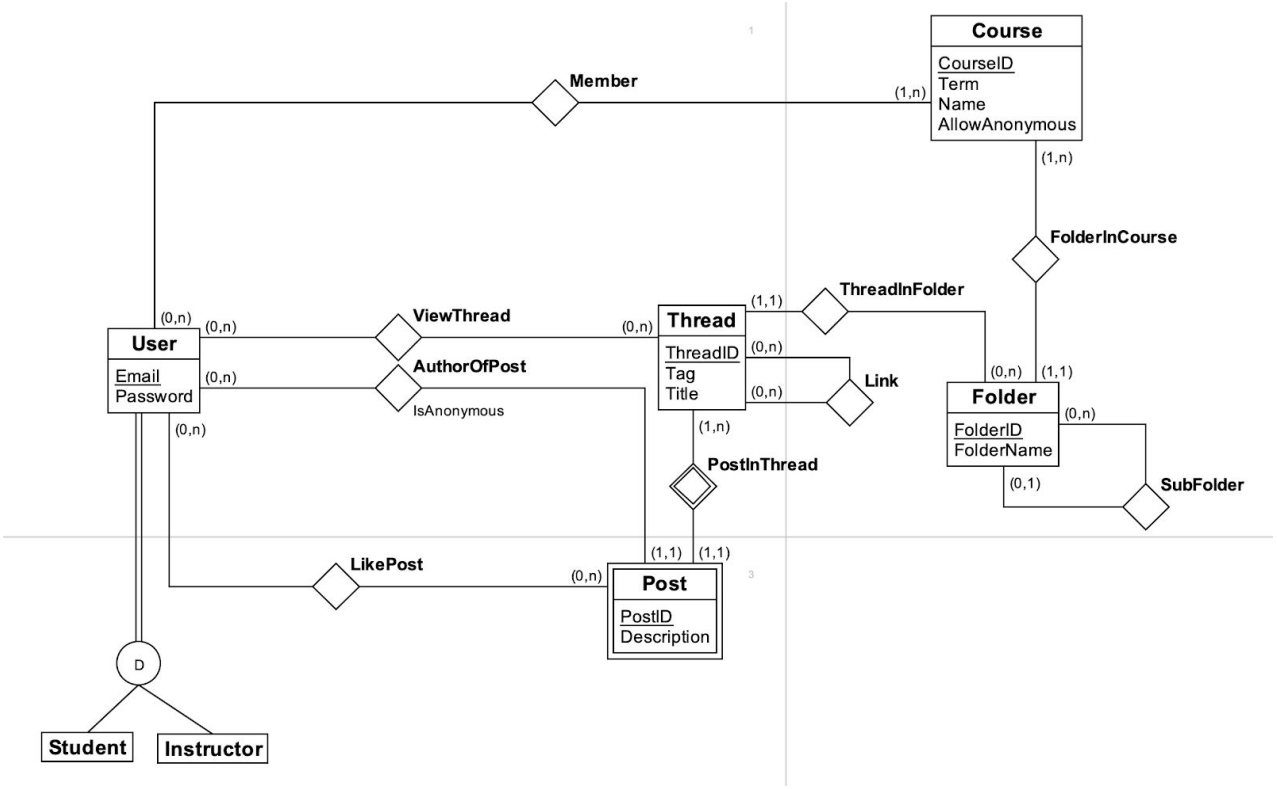
Prosjekt TDT4145

DB1: Gruppe 208

Silje Anfindsen, Vilde Haugsbakken Heggen og Torbjørn Syvertsen Stakvik

ER- modell:



Antagelser og forklaring av ER-modell:

For systemet over har vi laget en del egne antagelser basert på vår tolkning av prosjektbeskrivelsen og hvordan Piazza fungerer/burde fungere.

USER & COURSE

- Antar at det finnes to roller for brukere av Piazza: Instructor og Student. (Her er professor synonymt med instructor).
- Antar at det ikke er mulig å endre Email i systemet ettersom vi antar at det kun brukes ntnu-mail. Email fungerer også som brukernavn.
    - Dersom vi hadde inkludert en UserID som generert primærnøkkel i tillegg til Email og Password i User-tabellen, vil den funksjonelle avhengigheten Email->Password skape problemer for 3NF ettersom vi har en FA mellom to non-primes.

- Antar at en User kan eksistere i systemet uten relasjon til Course, men et Course krever minst en relasjon til User ettersom det skal være mulig for en Instructor å lage en bruker i Piazza og deretter opprette et Course.
    - Det er heller ikke spesifisert hvorvidt en Student får tilgang til et Course via invitasjon i ER-modellen, men dette vil bli implementert i applikasjonen. Begrunnelsen for at dette ikke tas med i modellen, er at det ikke er interessant å lagre hvem som har invitert hvem i databasen.

COURSE & FOLDER

- Antar at Course må ha minst en folder for å eksistere.
- Antar at Folder kun kan ha en ParentFolder, mens en ParentFolder kan ha flere ChildFolder.
- Antar at FolderID er unikt uavhengig av kurs.
    - Alternativt så vi her på å lage en svak klasse for Folder mot Course. Dette skapte problemer med normalformer i tabellene pga. SubFolder. Løsningen ble derfor heller basert på å generere en unik nøkkel for Folder, altså FolderID. Dette kan være en svakhet i et stort system med flere Course.

THREAD & POST

- Antar at hver gang det opprettes en Thread må det opprettes en tilhørende Post, altså en første Post i Thread.
    - En Thread i seg selv består ikke av Description fordi at idet det opprettes en Thread vil man kreve at det også opprettes en Post av samme User. Vår definisjon av Thread er at det fungerer som rammen rundt posten(e) bestående av en Tittel, Tag og tilhørende Folder. Post fungerer derfor både som første innlegg og som replies/kommentarer på innlegg. Vi har ikke tillatt flere threads / Follow-up-diskusjoner i en Thread.
- Post er modellert som svak entitetsklasse tilhørende Thread. PostID er altså kun unik for hver verdi av ThreadID.
- Vi antar at en Thread kan få color basert på relasjonene PostInThread og UserType.
    - Dersom det finnes mer enn én relasjon PostInThread (altså om den har en reply på første Post i Thread) sjekkes UserType hvor PostID >= 2. Dersom det for en av postene med PostID >= 2, finnes tilhørende bruker i AuthorOfPost hvor UserType er Instructor, får Thread en bestemt farge. Hvis ikke, får den en annen farge som indikerer at kun studenter har svart. Dersom det ikke finnes noen poster under Thread med PostID >= 2, velger vi å gi Thread en annen bestemt farge, som indikasjon på at ingen har svart.
- Antar at en bruker ser alle Post i en Thread. Vi skiller altså ikke mellom hvilke Post i en Thread en User kan se, derfor har vi kun relasjonsklassen ViewThread.
    - Dette påvirker også statistikken slik at Instructor kun kan se hvor mange Thread en User har sett.
- Vi lagrer ikke Search i databasen ettersom dette ikke er nødvendig for statistikk. Dette implementeres i applikasjonen.
- Antar at Link kun ser etter likhet mellom Threads ved å se på Tag og Title. Vi linker altså kun fra Thread til Thread og ikke til en konkret Post i en Thread.

Relasjonstabeller for modellen:

User(Email, Password, UserType)
Email fungerer som naturlig primærnøkkel.
UserType må ha en av verdiene Student eller Instructor.

Course(CourseID, Term, Name, AllowAnonymous)
CourseID er en generert primærnøkkel.
AllowAnonymous kan ha verdiene True eller False.

Member(Email, CourseID)
Koblingstabell. Email er fremmednøkkel mot User og CourseID er fremmednøkkel mot Course.

Folder(FolderID, FolderName, CourseID, ParentFolderID)
FolderID er en generert primærnøkkel.
CourseID er fremmednøkkel mot Course og kan ikke være NULL.
FolderName kan ta verdier; exam, exercise 1, etc.
ParentFolderID er fremmednøkkel mot Folder og kan være NULL.

Thread(ThreadID, Tag, Title, FolderID)
ThreadID er en generert primærnøkkel
FolderID er fremmednøkkel mot Folder og kan ikke være NULL.
Tag kan ta verdiene; Questions, Announcements, Homework, Homework solutions, Lectures notes eller General announcements.

ViewThread(Email, ThreadID)
Koblingstabell. Email er fremmednøkkel mot User og ThreadID er fremmednøkkel mot Thread.

Post(PostID, ThreadID, Description, Email, isAnonymous)
PostID er en generert verdi. Denne er kun unik for hver ThreadID.
ThreadID er fremmednøkkel mot Thread og kan ikke være NULL.
(PostID, ThreadID) er primærnøkkel ettersom Post er modellert som en svak entitetsklasse.
Email er fremmednøkkel mot User og kan ikke være NULL.
isAnonymous kan ta verdiene True eller False.

LikePost(Email, PostID, ThreadID)
Koblingstabell. Email er fremmednøkkel mot User og (PostID, ThreadID) er fremmednøkkel mot Post.

Link(SimilarThreadID, ThreadID)
Koblingstabell. SimilarThreadID er fremmednøkkel mot Thread og ThreadID er fremmednøkkel mot Thread.

Tilfredsstillelse av normalformer:

User(<u>Email</u>, Password, UserType)
F = { Email -> Password, UserType}
1NF: Alle attributter er atomiske.
2NF: Ingen delvis avhengigheter, så kravet er oppfylt.
3NF: Ingen transitive avhengigheter.
BCNF: Alle funksjonelle avhengigheter går ut fra supernøkkel Email.
4NF: Ingen MVD-er, så kravet er oppfylt.

Course(<u>CourseID</u>, Term, Name, AllowAnonymous)
F = { CourseID -> Term, Name, AllowAnonymous}
1NF: Alle attributter er atomiske.
2NF: Ingen delvise avhengigheter
3NF: CourseID er kandidatnøkkel, finnes derfor ingen transitive avhengigheter
BCNF: Alle FA går ut fra supernøkkel.
4NF: Alle FA går ut fra supernøkkel og alle ikke-nøkkelattributter er uavhengige av hverandre.

Member(<u>Email</u>, <u>CourseID</u>)
F = { Ingen }
1NF: Alle attributter er atomiske.
2NF: Det finnes ingen delvise avhengigheter i tabellen.
3NF: Det er ingen funksjonelle avhengigheter i tabellen.
BCNF: Det er ingen funksjonelle avhengigheter i tabellen.
4NF: Det er ingen MVD i tabellen fordi det er ingen FA i tabellen.

Folder(<u>FolderID</u>, FolderName, CourseID, ParentFolderID)
F = { FolderID -> FolderName, CourseID, ParentFolderID}
1NF: Alle attributter er atomiske.
2NF: Ingen delvis avhengigheter.
3NF: Alle FA går ut fra supernøkkel FolderID.
BCNF: Alle FA går ut fra supernøkkel FolderID.
4NF: Det er ingen MVD i tabellen.

Thread(<u>ThreadID</u>, Title, Tag, FolderID)
F = { ThreadID -> Title, Tag, FolderID }
1NF: Alle attributter er atomiske (alle attributter er udelelige).
2NF: Det finnes ingen delvis avhengigheter i tabellen.
3NF: Alle FA går ut fra supernøkkel ThreadID
BCNF: Alle FA går ut fra supernøkkel ThreadID
4NF: Det er ingen MVD i tabellen.

ViewThread(<u>Email, ThreadID</u>)
F = { Ingen }
1NF: Både Email og ThreadID er atomiske.
2NF: Ingen delvis avhengigheter i tabellen.

3NF: Det er ingen funksjonelle avhengigheter i tabellen.
BCNF: Det er ingen funksjonelle avhengigheter i tabellen.
4NF: Det er ingen MVD i tabellen fordi det er ingen FA.

Post(PostID, ThreadID, Description, Email, isAnonymous)
F = { (PostID, ThreadID) -> Description, Email, isAnonymous }
1NF: Alle attributter er atomiske (alle attributter er udelelige).
2NF: Det finnes ingen delvis avhengigheter i tabellen.
3NF: Alle avhengighetene er fullstendige, så ingen er transitive.
BCNF: Alle FA går ut fra supernøkkel (PostID, ThreadID).
4NF: Det er ingen MVD i tabellen.

LikePost(Email, PostID, ThreadID)
F = { Ingen }
1NF: Alle attributter er atomiske.
2NF: Ingen delvis avhengigheter i tabellen.
3NF: Det er ingen funksjonelle avhengigheter i tabellen.
BCNF: Det er ingen funksjonelle avhengigheter i tabellen.
4NF: Det er ingen MVD i tabellen.

Link(SimilarThreadID, ThreadID)
F = { Ingen }
1NF: Alle attributter er atomiske.
2NF: Ingen delvis avhengigheter i tabellen.
3NF: Det er ingen funksjonelle avhengigheter i tabellen.
BCNF: Det er ingen funksjonelle avhengigheter i tabellen.
4NF: Det er ingen MVD i tabellen.

Tilfredsstillelse av usecases:

1. A student logs into the system, i.e., check user name and password (you do not need to encrypt/decrypt passwords). This should have e-mail and password as input, and these should match this info in the database.

   Mål: Valider innlogging av bruker i systemet.
   For å sjekke om brukeren skriver riktig informasjon (inputEmail, inputPassword) kan man sjekke at det er nøyaktig 1 rad som returneres når spørringen er `SELECT * FROM User WHERE Email = inputEmail AND Password = inputPassword`.
   Hvis 0 rader returneres er det enten feil mail eller passord, hvis >1 rader returneres er det flere brukere med samme mail og passord, som ikke skal skje.

2. A student makes a post belonging to the folder “Exam” and tagged with “Question”. Input to the use case should be a post and the texts “Exam” and “Question”.

   Mål: Det må lages en Thread, og en tilhørende Post.

Programmet må finne ut hva Thread-ID skal være:
`SELECT ThreadID FROM Thread WHERE max(ThreadID)`
Les ThreadID fra den raden. La oss si i dette eksemplet at ThreadID = 64
Vi kan så opprette en ny Thread:
`INSERT INTO Thread (ThreadID, Tag, Title, FolderID) VALUES (65, "Question", “Title”, "Exam")`
Deretter opprettes en tilhørende Post:
`INSERT INTO Post(PostID, ThreadID, Description, Email, isAnonymous) VALUES (1, 65, "Description", "example@mail.com", false)`
Vi setter PostID til 1, siden dette er første Post tilhørende denne Thread’en.

3. An instructor replies to a post belonging to the folder “Exam”. The input to this is the id of the post replied to. This could be the post created in use case 2.

   Mål: Lager en ny Post i Thread’en i forrige oppgave.
   Finner verdi for PostID ved å telle antall Post’er som finnes under Thread’en fra før.
   La oss si det er 4 poster fra før under Thread’en.
   `INSERT INTO Post(PostID, ThreadID, Description, Email, isAnonymous) VALUES (5, 65, "Svar på spørsmål", "example2@mail.com", false)`

4. A student searches for posts with a specific keyword “WAL”. The return value of this should be a list of ids of posts matching the keyword.

   Mål: Søk etter match for et keyword i alle databasens poster uavhengig av tilhørende thread. Dersom keyword er nevnt i description minst en gang i minst en post i en thread får man match. Bruker får returnert eventuell postID med tilhørende ThreadID for hvert match.
   Bruker spørringen:
   `SELECT PostID, ThreadID From Post WHERE Description LIKE %WAL%`

5. An instructor views statistics for users and how many post they have read and how many they have created. These should be sorted on highest read posting numbers. The output is “user name, number of posts read, number of posts created”. You don’t need to order by posts created, but the number should be displayed. The result should also include users which have not read or created posts.

   Mål: Hent statistikk for antall poster brukere i systemet har laget og lest sortert etter brukere med høyest antall leste poster.
   Her er først nødvendig å sjekke UserType for å validere at det er instructor som ønsker å sjekke statistikk. Dette hentes ut fra User-tabellen under UserType-attributtet.
   `SELECT UserType FROM User WHERE Email = "example3@mail.com"`
   Hvis outputen av koden over er Instructor kan tilgangen til statistikken tillates.
   En kan deretter bruke en slik spørring i MySQL for å hente ut statistikk:
   `SELECT Email, COUNT(ThreadID) AS NumberRead, COUNT(PostID) AS NumberPosted`

```
FROM (User LEFT OUTER JOIN Post USING Email) LEFT OUTER JOIN
ViewThread USING Email
GROUP BY Email
SORT BY NumberRead DESC
```
(`NumberPosted` inneholder her summen av antall Thread startet av brukeren og antall besvarelser brukeren har gjort.)

SQL-kode for opprettelse av tabellene:

(Vedlagt i prosjekt.txt.)

```
use db1;

create table User(
Email varchar(30) not null,
Password varchar(30) not null,
UserType varchar(10) not null,
constraint user_PK primary key (Email),
constraint participants_ibfk CHECK (UserType IN ('Student','Instructor'))
);

create table Course(
CourseID int not null,
Name varchar(30),
Term varchar(10),
AllowAnonymous bool,
constraint course_PK primary key (CourseID)
);

create table Folder(
FolderID int not null,
FolderName varchar(30),
CourseID int not null,
ParentFolderID int,
constraint folder_PK primary key (FolderID),
constraint folder_FK1 foreign key (CourseID) references Course(CourseID)
on update cascade on delete cascade,
constraint folder_FK2 foreign key (ParentFolderID) references
Folder(FolderID) on update cascade on delete cascade
);

create table Member(
Email varchar(30) not null,
CourseID int not null,
constraint Member_PK primary key (Email, CourseID),
```

```
constraint Member_FK1 foreign key (Email) references User(Email) on update
cascade on delete cascade,
constraint Member_FK2 foreign key (CourseID) references Course(CourseID)
on update cascade on delete cascade
);

create table Thread(
ThreadID int not null,
Tag varchar(25),
Title varchar(30),
FolderID int not null,
constraint Thread_PK primary key (ThreadID),
constraint Thread_FK foreign key (FolderID) references Folder(FolderID) on
update cascade on delete cascade,
constraint participants_ibfk2 CHECK (Tag IN ('Questions', 'Announcements',
'Homework', 'Homework solutions', 'Lectures notes', 'General
announcements'))
);

create table ViewThread(
Email varchar(30) not null,
ThreadID int not null,
constraint ViewThread_PK primary key (Email, ThreadID),
constraint ViewThread_FK1 foreign key (Email) references User(Email) on
update cascade on delete cascade,
constraint ViewThread_FK2 foreign key (ThreadID) references
Thread(ThreadID) on update cascade on delete cascade
);

create table Post(
PostID int not null,
Description varchar(500),
Email varchar(30) not null,
isAnonymous bool,
ThreadID int not null,
constraint Post_PK primary key (PostID),
constraint Post_FK1 foreign key (Email) references User(Email) on update
cascade on delete cascade,
constraint Post_FK2 foreign key (ThreadID) references Thread(ThreadID) on
update cascade on delete cascade
);

create table LikePost(
PostID int not null,
Email varchar(30) not null,
constraint LikePost_PK primary key (PostID, Email),
```

```
constraint LikePost_FK1 foreign key (PostID) references Post(PostID) on
update cascade on delete cascade,
constraint LikePost_FK2 foreign key (Email) references User(Email) on
update cascade on delete cascade
);

create table Link(
ThreadID int not null,
SimilarThreadID int not null,
constraint Link_PK primary key (ThreadID, SimilarThreadID),
constraint Link_FK1 foreign key (ThreadID) references Thread(ThreadID) on
update cascade on delete cascade,
constraint Link_FK2 foreign key (SimilarThreadID) references
Thread(ThreadID) on update cascade on delete cascade
);
```