Normalized tables:

We choose to normalize following the rules from BCNF

Steps:

1. Normalizing Users Table:

Original:Users(Id:INT [PK], Email:VARCHAR(255) [UNIQUE NOT NULL],
Password:VARCHAR(255) [NOT NULL],
Schools:JSON [FK to Schools.Name], FinalSchool:VARCHAR(255) [FK to Schools.Name])

Since a user may have interest in multiple schools, we separate this attribute into its own table:

```
Users(
        Id INT PRIMARY KEY,
        Email VARCHAR(255) UNIQUE NOT NULL,
        Password VARCHAR(255) NOT NULL,
        FinalSchool VARCHAR(255)
)

User_Schools(
                User_id INT NOT NULL [FK to Users.Id],
                SchoolId INT NOT NULL  [FK to Schools.school_id],
                PRIMARY KEY (User_id, SchoolId)
)
```

2. Normalizing NewsFeed Table:

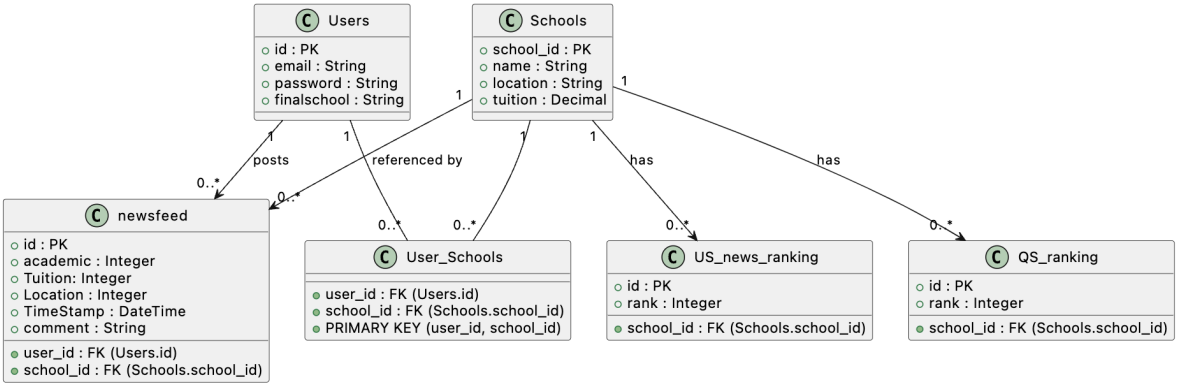
Original: NewsFeed(Id:INT [PK], Email:VARCHAR(255) [FK to Users.Email NOT NULL], User_id:INT [FK to Users.Id NOT NULL], School_id:INT [FK to Schools.School_id NOT NULL], Academic:INT [DEFAULT 0 NOT NULL], Tuition:INT [DEFAULT 0 NOT NULL], Location:INT [DEFAULT 0 NOT NULL], Comment:VARCHAR(MAX), Timestamp:DATETIME [NOT NULL])

We can see that the email is redundant, we can identify the user just by User_id, so we will remove that.

```
NewsFeed(
        Id:INT [PK],
        User_id:INT [FK to Users.Id NOT NULL],
        School_id:INT [FK to Schools.School_id NOT NULL],
        Academic:INT [DEFAULT 0 NOT NULL],
        Tuition:INT [DEFAULT 0 NOT NULL],
        Location:INT [DEFAULT 0 NOT NULL],
        Comment:VARCHAR(MAX),
        Timestamp:DATETIME [NOT NULL]
        )
```

All other tables are simple enough, no further normalizations needed.

UML Diagram:



Relationship and Cardinality

Users and User_Schools: one-many

> One user can be interested in many different schools, so User can be associated with multiple School. Also, `User_Schools` achieved a many-many relationship between Users and Schools as it connects the two together.

Users and NewsFeed: one-many

> One User can create many different NewsFeed posts. A `User_id` in `NewsFeed` references `Users.Id`, which makes this a one-many relationship.

Schools and User_Schools: one-many

> One School can have multiple Users associated with it through `User_Schools`. Again, this contributes to the many-many relationship between Users and Schools.

Schools and NewsFeed: one-many

One School can have multiple NewsFeed related to it. The `School_id` in `NewsFeed` references `Schools.School_id`, which makes this a one-many relationship.

Schools and US_NEWS_RANKING: one-one

Each School has only one ranking in `US_NEWS_RANKING`. Since `School_id` in `US_NEWS_RANKING` is `UNIQUE`, it ensures a one-one relationship.

Schools and QS_RANKING: one-one

Similar to US_NEWS_RANKING. Each School has only one ranking in `QS_RANKING`. The `School_id` in `QS_RANKING` is `UNIQUE`, enforcing a one-one relationship.

Assumptions for Entity and Relationship

Users Table

The users table represents individuals who interact with the site.
Id: Primary Key for each user.
Email: Each user can be uniquely identified by their email. The email should be unique and have no duplications and not null.
Password: The password field stores a hashed password by jwt, and can not be null. The structure of the password (such as length requirement and complexity requirements) are enforced through the front end.
FinalSchool: This field holds the user's selected final school. It is intended to reference a school from the Schools table

User_Schools Table

This table captures the many_to_many relationship between users and the schools they are interested in.

User_id: Foreign key referencing Users.Id.

SchoolId: Foreign key referencing Schools.school_id.

Primary Key: The composite key (User_id, SchoolId) ensures that each association between a user and a school is unique.

NewsFeed Table

The newsfeed table captures the posts and its content that appear on the news feed page.
Email: This field links each newsfeed post to the user who created it, and it is a foreign key to the Users table's email column. This integrity is also enforced by rejecting modifications. This can not be NULL.
Id: Primary Key for each newsfeed post
User_id: Represents the user the post is associated with. It is stored as an int field, and as a foreign key to the users table's Users.id column. The integrity is enforced by rejecting modifications. This can not be NULL.
School_id: Represents the school the post is associated with. It is stored as an int field, and as a foreign key to the School table's school_idcolumn. The integrity is enforced by rejecting modifications. This can not be NULL.
Academic, Tuition, Location: These integer fields represent ratings out of 5 that provide quantitative evaluations of a school in specific areas. Those integers will turn into stars in the front end. This field will default to 0, which is not NULL.
Comment: The comment field holds the post content, this field can be NULL.
Timestamp: we store a time variable so that the frontend can sort the news in latest to oldest order. This does not have to be unique, but has to be NOT NULL.

Schools Table

The Schools table is designed as a dedicated repository for storing essential information about each educational institution. We assume that every school must have a valid name, as well as valid location and tuition information.
School_id: primary key for each school
Name: A UNIQUE NOT NULL attribute that represents the school's name
Location: School's location
Tuition: School's tuition

US_NEWS_RANKING Table

This table holds reference data imported and cleaned from US-NEWS, providing details about various colleges. This is an independent entity because US news ranking is independent from QS, and it contains more information such as location and tuition which is hard to combine with the QS table. We assume that Name is unique and not null, and shall be referencing the main Schools table.
Id: Primary Key for each us news ranking entry
Name: Schools name, which is UNIQUE and NOT NULL, and a FOREIGN KEY to Schools.Name.
Rank: School's US News Ranking, NOT NULL

QS_RANKING Table

This table holds reference data imported and cleaned from QS, providing details about various colleges. We assume that Name is unique and not null, and shall be referencing the main Schools table, while rank QS can't be null.
Id: Primary Key for each QS ranking entry
Name: Schools name, which is UNIQUE and NOT NULL, and a FOREIGN KEY to Schools.Name.
Rank: School's US News Ranking, NOT NULL

Logical Table:

```
Users(
        Id INT PRIMARY KEY,
        Email VARCHAR(255) UNIQUE NOT NULL,
        Password VARCHAR(255) NOT NULL,
        FinalSchool VARCHAR(255)
)

User_Schools(
        User_id INT NOT NULL [FK to Users.Id],
        SchoolId INT NOT NULL  [FK to Schools.school_id],
        PRIMARY KEY (User_id, SchoolId)
)

NewsFeed(
        Id:INT [PK],
        User_id:INT [FK to Users.Id NOT NULL],
        School_id:INT [FK to Schools.School_id NOT NULL],
        Academic:INT [DEFAULT 0 NOT NULL],
        Tuition:INT [DEFAULT 0 NOT NULL],
        Location:INT [DEFAULT 0 NOT NULL],
        Comment:VARCHAR(MAX),
        Timestamp:DATETIME [NOT NULL]
)

Schools(
        School_id:INT [PK],
```

```
        Name:VARCHAR(255) [UNIQUE NOT NULL],
        Location:VARCHAR(255) [NOT NULL],
        Tuition:DECIMAL [NOT NULL]
)

US_NEWS_RANKING(
        Id:INT [PK],
        School_id:INT [FK to Schools.School_id UNIQUE NOT NULL],
        Rank:INT [NOT NULL]
)

QS_RANKING(
        Id:INT [PK],
        School_id:INT [FK to Schools.School_id UNIQUE NOT NULL],
        Rank:INT [NOT NULL]
)
```