1. Design your collection schemas

User:

```
const User = new mongoose.Schema({
        name:{
                type: String,
                required:true
        },
        joined:{
                type: Date ,
                required:true
        }
});
```

Post:

```
const Post = new mongoose.Schema({
        author:{
                type: String,
                required:true,
                index: true
        },
        content:{
                type: String,
                required:true
        },
        created:{
                type:Date,
                required:true,
                index:true
        }
},
{
        timestamps: true,
});
```

Follows:

```
const Follow = new mongoose.Schema({
        follower:{
            type: mongoose.Schema.Types.ObjectId,
            ref: "User",
            required: true,
```

```
        index:true
    },
    following:{
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
        required: true,
        index:true
    },
    created:{
            type:Date,
            required:true,
            index:true
    }

});
```

Aggregation Queries

a. **List each user followers**

```
aggregate([{
 $match:{
  following:ObjectId('68755260cfbe1fc6f54fbb0c')
 },
},{
        $lookup: {
         from: "users",
         localField: "follower",
         foreignField: "_id",
         as: "user"
        }
},
{
  $unwind: {
   path: "$user",
   preserveNullAndEmptyArrays: true
  }
 },
{
  $project: {
   "_id":"$user._id",
   "name":"$user.name",
   "joined":"$user.joined"
  }
```

```
 }
]);
```

b. List each user Follows

```
[{
 $match:{
  follower:ObjectId('68755064cfbe1fc6f54fbafe')
 },
},{
          $lookup: {
           from: "users",
           localField: "following",
           foreignField: "_id",
           as: "user"
          }
},
{
  $unwind: {
    path: "$user",
    preserveNullAndEmptyArrays: true
  }
 },
 {
  $project: {
    "_id":"$user._id",
    "name":"$user.name",
    "joined":"$user.joined"
  }
 }
]
```

c. Page through post in chronological order

```
[{
 $match: {
  author:ObjectId('68755260cfbe1fc6f54fbb0c')
 }
},{
 $sort: {
  created: -1
 }
},{
 $skip: 0,
},{
 $limit: 10
}]
```

2. Aggregation Pipeline – for a given user ID

```
aggregate([
 {
  $lookup: {
   from: "follower",
   localField: "author",
   foreignField: "following",
   as: "follow_relationship"
  }
 },
 {
  $match: {
   "follow_relationship.follower": ObjectId('68755064cfbe1fc6f54fbafe')
  }
 },
 {
  $lookup: {
   from: "users",
   localField: "author",
   foreignField: "_id",
   as: "author"
  }
 },
 {
  $unwind: {
   path: "$author"
  }
 },
 {
  $sort: {
   created: -1
  }
 },
 {
  $project: {
   _id:1,
   author:"$author.name",
   content:1,
   created:1
```

```
    }
  }
]);
```

3. Index Collection

In **Post collection** Index are create on author and created to speed up record retrieval when listing a user’s post in chronological order
Also in **follows collection** Indexes are created on the follower, following and created fields to speed up record retrieval when listing a user's followers or the users they follow.