Database Design

DDL Commands

```
CREATE TABLE User (
    userID INT PRIMARY KEY,
    userName VARCHAR(255),
    email VARCHAR(255),
    universityID INT,
    FOREIGN KEY (universityID) REFERENCES University(universityID)
);

CREATE TABLE Family (
    familyID INT PRIMARY KEY,
    leaderID INT,
    accessType VARCHAR(255),
    serviceName VARCHAR(255),
    FOREIGN KEY (leaderID) REFERENCES User(userID),
    FOREIGN KEY (serviceName) REFERENCES SubscriptionService(serviceName)
);

CREATE TABLE SubscriptionService (
    serviceName VARCHAR(255) PRIMARY KEY,
    price DECIMAL(10,2),
    maxMembers INT
);

CREATE TABLE University (
    universityID INT PRIMARY KEY,
    universityName VARCHAR(255),
    city VARCHAR(255)
);

CREATE TABLE Membership (
    memberID INT,
    familyID INT,
    memberStatus VARCHAR(255),
    PRIMARY KEY (memberID, familyID),
    FOREIGN KEY (memberID) REFERENCES User(userID),
    FOREIGN KEY (familyID) REFERENCES Family(familyID)
);

CREATE TABLE BankAccount (
    accountName VARCHAR(255),
    platform VARCHAR(255),
    userID INT,
    PRIMARY KEY (accountName, platform),
    FOREIGN KEY (userID) REFERENCES User(userID)
);
```

```
CREATE TABLE Payment (
    paymentID INT PRIMARY KEY,
    payerID INT,
    recipientID INT,
    amount DECIMAL(10,2),
    paid BIT,
    deadline DATE,
    FOREIGN KEY (payerID) REFERENCES User(userID),
    FOREIGN KEY (recipientID) REFERENCES User(userID)
);
```

Advanced Queries

Query 1

Our first query returns the number of pending user invitations per university. We first join the `University` table with the `User` table, then group by the `universityID`. We order the outputby descending number of users.

Code

```
select un.universityName, count(*) as numUsers
from University un
natural join User us
group by un.universityID
order by numUsers desc;
```

Result

```python
with pool.connect() as con:
    statement = '''
    select un.universityName, count(*) as numPending
    from University un
    natural join User us
    join Membership m
    on m.memberID = us.userID
    where m.memberStatus = "Pending"
    group by un.universityName
    order by numPending desc;
    '''
    result = con.execute(statement).fetchall()
    for row in result:
        print(row)
```

[123] ✓ 0.9s Python

```
... ('Harper College', 73)
    ('Northwestern University', 61)
    ('Yale University', 61)
    ('University of Illinois at Urbana-Champaign', 60)
    ('Harvard University', 57)
    ('Massachusetts Institute of Technology', 57)
```

Initial Performance

Below is the performance of our query without any added indexes.

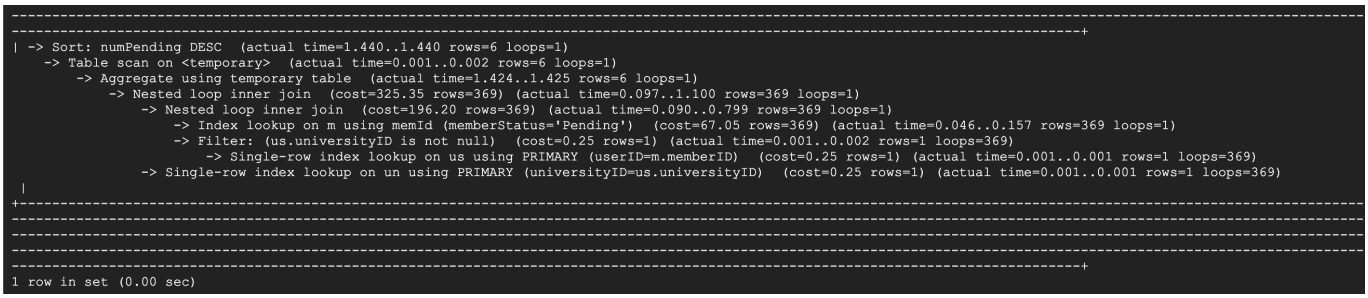
```
| -> Sort: numPending DESC  (actual time=3.004..3.004 rows=6 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.001 rows=6 loops=1)
        -> Aggregate using temporary table  (actual time=2.958..2.959 rows=6 loops=1)
            -> Nested loop inner join  (cost=683.50 rows=400) (actual time=0.388..2.652 rows=369 loops=1)
                -> Nested loop inner join  (cost=543.50 rows=400) (actual time=0.380..2.345 rows=369 loops=1)
                    -> Filter: (m.memberStatus = 'Pending')  (cost=403.50 rows=400) (actual time=0.365..1.670 rows=369 loops=1)
                        -> Table scan on m  (cost=403.50 rows=4000) (actual time=0.358..1.329 rows=4000 loops=1)
                    -> Filter: (us.universityID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=369)
                        -> Single-row index lookup on us using PRIMARY (userID=m.memberID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=369)
                -> Single-row index lookup on un using PRIMARY (universityID=us.universityID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=369)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
---+
1 row in set (0.01 sec)
```

Index 1

For our first index design, we added an index on `Membership.memberStatus` because we filter on that column in the query.

```sql
alter table Membership add index memId(memberStatus);
```

The output of our analyze query follows:



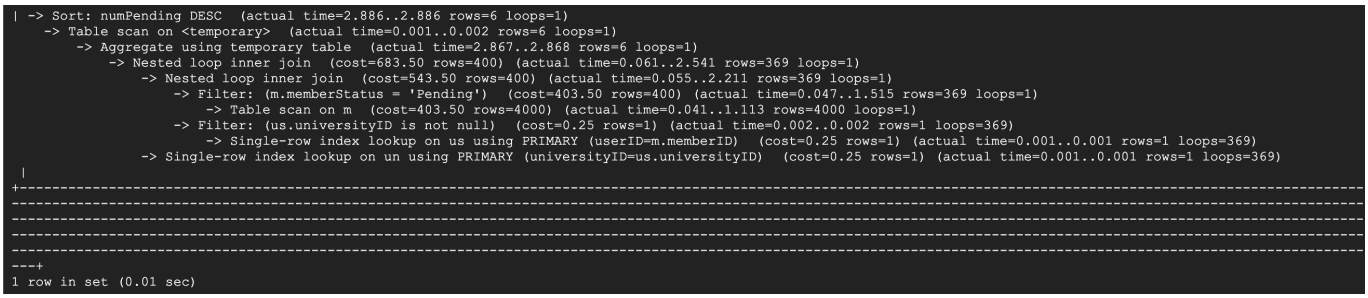
We see a speedup in the "Filter" command that addresses `m.memberStatus = "Pending"`, and an overall speedup in the whole query, so we can conclude that this index is a good optimization to add for this specific query.

Index 2

For our second index design, we added an index on `University.universityName` because we use it in the grouping step of our query.

```
alter table University add index uniName(universityName);
```

The output of our analyze query follows:



Query 2

Our second query returns the number of accepted users for every university and subscription service. We begin by joining the `User`, `Membership`, `Family`, `SubscriptionService`, and `University` tables. We then filter by accepted members, and group by both `universityID` and `serviceName`. We order the output by the `universityName` in ascending order and aggregated `numUsers` in descending order. Finally, we limit the length of the output to 15 rows for visualization.

Code

```
select un.universityName, ss.serviceName, count(*) as numUsers
from User us
join Membership m
    on m.memberID = us.userID
natural join Family f
natural join SubscriptionService ss
natural join University un
where m.memberStatus = "Accepted"
```

```
group by un.universityID, ss.serviceName
order by un.universityName asc, numUsers desc
limit 15;
```

Result

```
with pool.connect() as con:
    statement = '''
    select un.universityName, ss.serviceName, count(*) as numUsers
    from User us
    join Membership m
        on m.memberID = us.userID
    natural join Family f
    natural join SubscriptionService ss
    natural join University un
    where m.memberStatus = "Accepted"
    group by un.universityID, ss.serviceName
    order by un.universityName asc, numUsers desc
    limit 15;
    '''
    result = con.execute(statement).fetchall()
    for row in result:
        print(row)
```

[118] ✓ 0.1s Python

```
('Harper College', 'Netflix', 135)
('Harper College', 'Spotify', 131)
('Harper College', 'Amazon Prime Video', 126)
('Harper College', 'Hulu', 99)
('Harper College', 'Disney+', 90)
('Harvard University', 'Netflix', 107)
('Harvard University', 'Disney+', 106)
('Harvard University', 'Amazon Prime Video', 100)
('Harvard University', 'Spotify', 100)
('Harvard University', 'Hulu', 93)
('Massachusetts Institute of Technology', 'Spotify', 112)
('Massachusetts Institute of Technology', 'Amazon Prime Video', 104)
('Massachusetts Institute of Technology', 'Netflix', 103)
('Massachusetts Institute of Technology', 'Disney+', 93)
('Massachusetts Institute of Technology', 'Hulu', 93)
```

Indexing

For the second query, we tried another three index designs.