Database Design Project | Roommate Matching System for Students

Project Direction Overview:

We want to develop a website/app to help people find a roommate. When people use our app, they can find a roommate they like. It is hard for people to find a roommate if they move to a completely new place, especially for international students, since they have no friends there. We want to create a platform for those people to find their roommates. Typically, a student could create an account and input his personal information, including gender, age, school, major and so on. And he can find roommates by input some requirements they want, like the price and location of the apartment; the characteristics of the roommate, like gender, age, school. After this, we can provide him some information about other users and he can contact who he likes.

Use Cases and Fields:

Account Signup/Installation Use Case

1. The person visits Under-roof buddy' website or app store and installs theapplication.
2. The application asks them to create an account when it first runs.
3. The user enters their information for the profile and the account is created in thedatabase.

Field	What it Stores	Why it's Needed
AccountID	This field stores a summary name associated with each account,	Give users a pseudonym to protect their privacy.
Password	This is the password of the account	For users to log in and protect the account.

FirstName	This is the first name of account holder	This is necessary for displaying the person’s name on screens and addressing them when sending them emails or other communications.

MiddleName		
LastName	This is the last name of account holder	This is necessary for displaying the person’s name on screens and addressing them when sending them emails or other communications.
Gender	This is the gender of account holder	This item can help users filter roommates from the gender perspective.
Age	This is the age of account holder	This item can help users filter roommates from the perspective of age.
School	This is the gender of account holder	This item can help users select roommates from different schools.
Major	This is the age of account holder	This item can help users screen roommates from different majors.
Photo	This is the photo of account holder	This item stores users' photos to facilitate offline contact.

Nationality	This is the nationality of account holder	This item helps users choose roommates of different nationalities.
CheckinDate	This is the check in date of account holder	This item can help users find time matched roommates.
PhoneNumber	This is the telephone number of account holder	This item is convenient for users to be contacted by other qualified users by phone.
EmailAddress	This is the email address of account holder	This item is convenient for users to be contacted by other qualified users via email.
Budget	This is the user's budget for rent.	This item can help users find budget matched roommates.
Location	This is the user's expectation of location.	This item can help users find location matched roommates.
HouseType	This is the user's expectation of the house type.	This item can help users find house type matched roommates.

Another use case is for users to input their expectation of apartment and roommate
Matching Roommates Use Case

Users can see who reaches their expectation by inputting the requirements.
Significant fields are detailed below.

Field	What it Stores	Why it’s Needed

Budget	This is the user's expectation of rent.	Help users screen out roommates that fit their budget.
Location	This is the user's expectation of location.	Help users match roommates who want to rent in the same city and area.
HouseType	This is the user's expectation of the house type.	Help users match roommates in the same apartment but in different rooms.
Gender	This is the expectation of gender of account holder	Help users match roommates of the specified gender.
Age	This is the expectation of age of account holder	Help users match roommates of a specified age group.
School	This is the expectation of school of account holder	Help users match the roommates of the specified school.
Major	This is the expectation of major of account holder	Help users to match their roommates in the specified specialty.
Nationality	This is the expectation of nationality of account holder	Help users match their roommates with the specified nationality.
CheckInDate	This is the expectation of check in date of account holder	Help users match their roommates with the specified nationality.

Another important usage is to allow users to communicate with candidates directly.

Communication Use Case

1. The users can get the photos, telephone number and email address ofCandidates.
2. The database would make use of the fields already highlighted for the second usecase.

Field	What it Stores	Why it’s Needed
FirstName	This is the first name of account holder	Get roommate’s information
MiddleName	This is the middle name of account holder	Get roommate’s information
LastName	This is the last name of account holder	Get roommate’s information
Photos	This is the photo of account holder	Get roommate’s information
Telephone number	This is the telephone number of account holder	Provide the telephone contact information of matching results for users.
Email address	This is the email address of account holder	Provide users with email contact information of matching results.

Case1:

1. The person visits Under-roof buddy' website or app store and installs theapplication.

2. The application asks them to create either a free or paid account when it first runs.Free accounts can only see who reaches their requirement but can not see the personal information and contact others.

3. The user selects the type of account and enters their information and the accountis created in the database.

Rule1. Each Account is associated with a profile; each profile is associated with one Account.

Rule2. An account is a free account or a paid account.

Case2:

Users can see who reaches their expectation by inputting the requirements.

Rule3. Each Account may have many requests, one request can be owned by many Accounts.

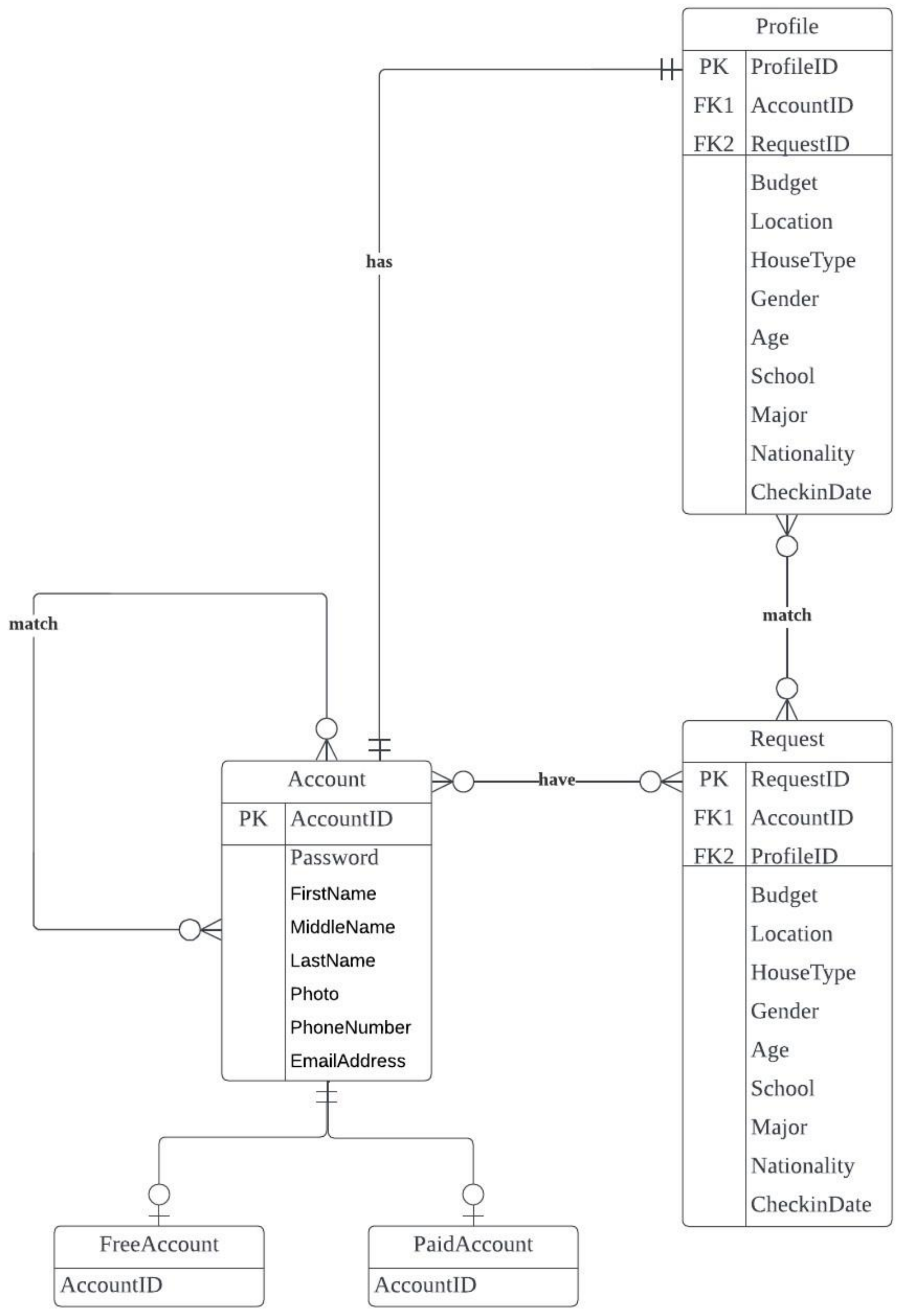
Rule4: Each profile can match many requests, each request can match many accounts.

Case3:

1. The users can get the photos, telephone number and email address ofCandidates.

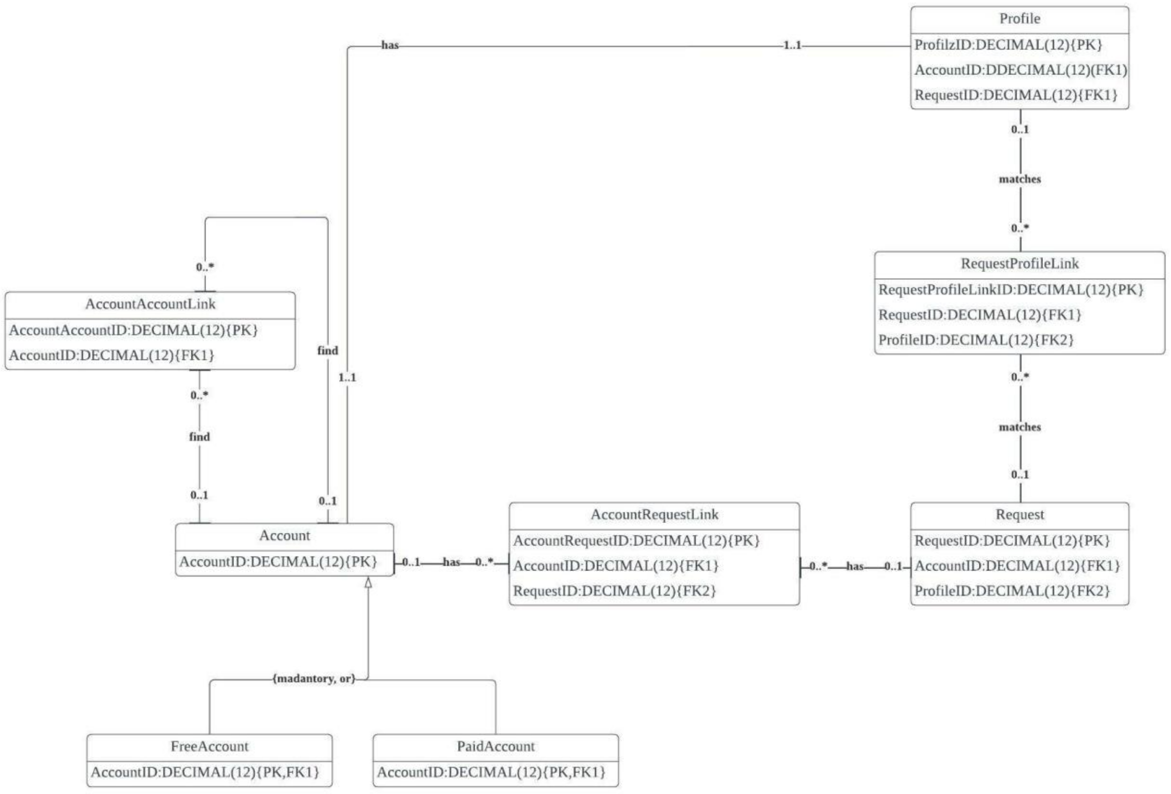
2. The database would make use of the fields already highlighted for the second usecase.

Rule5. One account may match many accounts, one account may be matched by many accounts.



We add specialization-generalization relationships to the Account.

We transfer the conceptual ERD(Crow's Foot) to the DBMS physical ERD(UML). Since Request/Profile and Account/Account are M:N relationships, we create two bridging entities, named RequestProfileLink and AccountAccountLink. They both have two 1:M relationships. We also make all primary keys of the DECIMAL(12) datatype.



Profile
ProfilzID:DECIMAL(12){PK}
AccountID:DDECIMAL(12)(FK1)
RequestID:DECIMAL(12){FK1}
has
1..1
0..1
matches
0..*
RequestProfileLink
RequestProfileLinkID:DECIMAL(12){PK}
RequestID:DECIMAL(12){FK1}
ProfileID:DECIMAL(12){FK2}
0..*
matches
0..1
AccountAccountLink
AccountAccountID:DECIMAL(12){PK}
AccountID:DECIMAL(12){FK1}
0..*
find
0..1
find
1..1
0..1
Account
AccountID:DECIMAL(12){PK}
0..1
has
0..*
AccountRequestLink
AccountRequestID:DECIMAL(12){PK}
AccountID:DECIMAL(12){FK1}
RequestID:DECIMAL(12){FK2}
0..*
has
0..1
Request
RequestID:DECIMAL(12){PK}
AccountID:DECIMAL(12){FK1}
ProfileID:DECIMAL(12){FK2}
{madantory, or}
FreeAccount
AccountID:DECIMAL(12){PK,FK1}
PaidAccount
AccountID:DECIMAL(12){PK,FK1}

ADD NEW

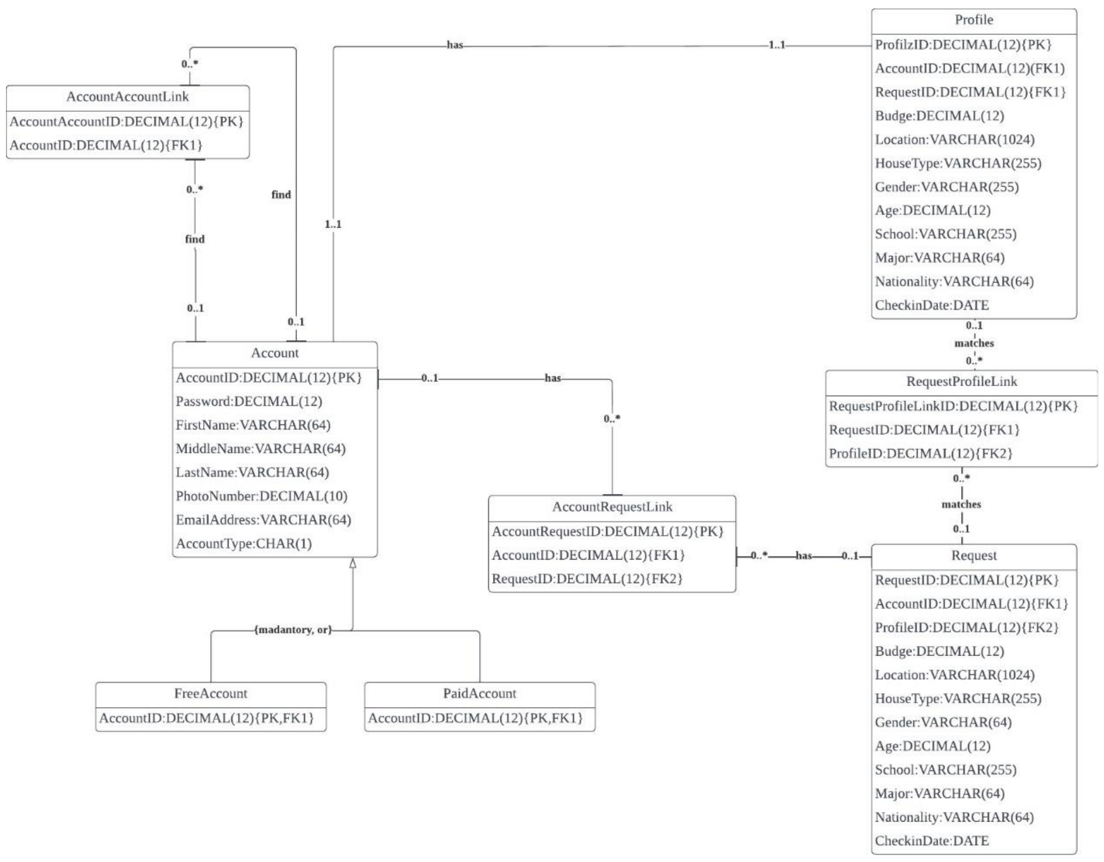
This is the new table and the final ERD.

Table	**Attribute**	**Datatype**	**Reasoning**
Account	Password	DECIMAL(12)	When people set a password to log in, the password is allowed to have 12 digits. 12 bits are allowed as the upper security limit.
Account	FirstName	VARCHAR(64)	This is the first name of the account holder, up to 64 characters of the name.

Account	MiddleName	VARCHAR(64)	This is the middle name of the account holder, up to 64 characters of the name.
Account	LastName	VARCHAR(64)	This is the last name of the account holder, up to 64 characters of the name.
Account	PhotoNumber	DECIMAL(10)	When people set the phone number of the account, the phone number is allowed to have 12 digits. 10 bits are allowed as the upper security limit.
Account	EmailAddress	VARCHAR(255)	Each user can set and request the desired location information. 1024 characters are allowed as the maximum security limit.
Account	AccountType	CHAR(1)	In a prior iteration, I indicated that there can be at least two types of account, free and paid. This attribute is the subtype discriminator to indicate which it is.
Profile	Budge	DECIMAL(6)	When people set the budget for rental demand, the budget price is allowed to have 6 digits. It is allowed to use 6 digits as the upper security limit.
Profile	Location	VARCHAR(1024)	Every product may have a description. People may want to describe the product more than just with the name. I allow for 1,024 characters so that people can type in something long if they need to.
Profile	HouseType	VARCHAR(255)	Each user can choose and request the type of room they expect. Use 255 characters as the safe upper limit.
Profile	Gender	VARCHAR(64)	Each user should have his/her own gender and use 64 bytes for storage.
Profile	Age	DECIMAL(2)	When people set their own age, the age number is allowed to have 2 digits. Two digits are

			allowed as the upper security limit.
Profile	School	VARCHAR(255)	Users may be students, so they can provide their school name for those who need to filter, and use 255 bytes for storage.
Profile	Major	VARCHAR(64)	Users may be students, so they can provide their own professional name for those who need to filter, and use 64 bytes for storage.
Profile	Nationality	VARCHAR(64)	Each user should have his own nationality and use 64 bytes for storage.
Profile	CheckinDate	DATE	Users can set the time they want to stay.
Request	Budge	DECIMAL(6)	When people set the budget for rental demand, the budget price is allowed to have 6 digits. It is allowed to use 6 digits as the upper security limit.
Request	Location	VARCHAR(1024)	Each user can set and request the desired location information. 1024 characters are allowed as the maximum security limit.
Request	HouseType	VARCHAR(255)	Each user can choose and request the type of room they expect. Use 255 characters as the safe upper limit.
Request	Gender	VARCHAR(64)	Each user can request and filter the gender of his roommate and store it in 64 bytes.
Request	Age	DECIMAL(2)	When people set their own age, the age number is allowed to have 2 digits. Two digits are allowed as the upper security limit.
Request	School	VARCHAR(64)	Users may be students, so they can provide their school name for those who need to filter, and use 255 bytes for storage.

Request	Major	VARCHAR(64)	Users may be students, so they can provide their own professional name for those who need to filter, and use 64 bytes for storage.
Request	Nationality	VARCHAR(64)	Each user should have his own nationality and use 64 bytes for storage.
Request	CheckinDate	DATE	Users can set the time they want to stay.



And we create tables in SQL, using the following code.

```
CREATE TABLE Account (

accountID DECIMAL(12) NOT NULL PRIMARY KEY,

password DECIMAL(12) NOT NULL,

firstName VARCHAR(64) NOT NULL,

lastName VARCHAR(64) NOT NULL,

phoneNumber DECIMAL(12) NOT NULL,

email VARCHAR(64) NOT NULL,

accountType CHAR(1) NOT NULL);


CREATE TABLE FreeAccount (

accountID DECIMAL(12) NOT NULL PRIMARY KEY,

FOREIGN KEY (accountID) REFERENCES Account(accountID));


CREATE TABLE PaidAccount (

accountID DECIMAL(12) NOT NULL PRIMARY KEY,

FOREIGN KEY (accountID) REFERENCES Account(accountID));


CREATE TABLE Profile (

profileID DECIMAL(12) NOT NULL PRIMARY KEY,

budget DECIMAL(12) NOT NULL,

location VARCHAR(255) NULL,

houseType VARCHAR(64) NULL,

gender VARCHAR(64) NOT NULL,

age DECIMAL(12) NOT NULL,

school VARCHAR(255) NOT NULL,

major VARCHAR(64) NOT NULL,

nation VARCHAR(64) NOT NULL,

checkInDate DATE NOT NULL,
accountID DECIMAL(12) NOT NULL,
FOREIGN KEY (accountID) REFERENCES Account(accountID));

CREATE TABLE Request (
requestID DECIMAL(12) NOT NULL PRIMARY KEY,
budget DECIMAL(12) NULL,
location VARCHAR(255) NULL,
houseType VARCHAR(64) NULL,
gender VARCHAR(64) NULL,
age DECIMAL(12) NULL,
school VARCHAR(255) NULL,
major VARCHAR(64) NULL,
nation VARCHAR(64) NULL,
checkInDate DATE NULL,
accountID DECIMAL(12) NOT NULL,
profileID DECIMAL(12) NOT NULL,
FOREIGN KEY (accountID) REFERENCES Account(accountID),
FOREIGN KEY (profileID) REFERENCES Profile(profileID));

CREATE TABLE AcountAcountLink (
accountAccountID DECIMAL(12) NOT NULL PRIMARY KEY,
accountID DECIMAL(12) NOT NULL,
FOREIGN KEY (accountID) REFERENCES Account(accountID));

CREATE TABLE AcountRequestLink (
accountRequestID DECIMAL(12) NOT NULL PRIMARY KEY,
accountID DECIMAL(12) NOT NULL,
requestID DECIMAL(12) NOT NULL,
FOREIGN KEY (accountID) REFERENCES Account(accountID),

FOREIGN KEY (requestID) REFERENCES Request(requestID));


CREATE TABLE RequestProfileLink (

requestProfileID DECIMAL(12) NOT NULL PRIMARY KEY,

requestID DECIMAL(12) NOT NULL,

profileID DECIMAL(12) NOT NULL,

FOREIGN KEY (requestID) REFERENCES Request(requestID),

FOREIGN KEY (profileID) REFERENCES Profile(profileID));
```

Normalization

DBMS Physical ERD above has already been normalized. The Account entity is normalized to BCNF because it doesn’t have partial dependencies and transitive dependencies, but it has only one candidate key–PhoneNumber. Also, Profile and Request are normalized to 3NF, since they don’t have partial dependencies and transitive dependencies.

Identifying Columns Needing Indexes for my Database

As far as primary keys which are already indexed, here is the list.

AccountID

RequestID

ProfileID

Below is a table identifying each foreign key column, whether or not the index should be unique or not, and why.

Column	Unique?	Description
AccountID	Not unique	The foreign key in Profile and Budget is not unique because many accounts could be matched by the same accounts.
RequestID	Not unique	The foreign key in Profile and Account is not unique because many persons’ profiles can be matched by the same person’s request.

ProfileID	Not unique	The foreign key in Request and Account is not unique because many persons' requests can be matched by the same person's profile.

As far as the three query driven indexes, I spotted three fairly easily by predicting what columns will commonly be queried. For example, it's reasonable that there will be many queries that are limited by budget to see whether people are able to afford the renting price. So I select the column budget to be indexed. This would be a non-unique index because many requests or profiles could have the same budgets.

It's also reasonable that the check-in date will be a limiting column in queries, because students will commonly move in during the same period, such as a particular month, week or date. So I select CheckinDate to index. This would be a non-unique index because many people would move in on the same day.

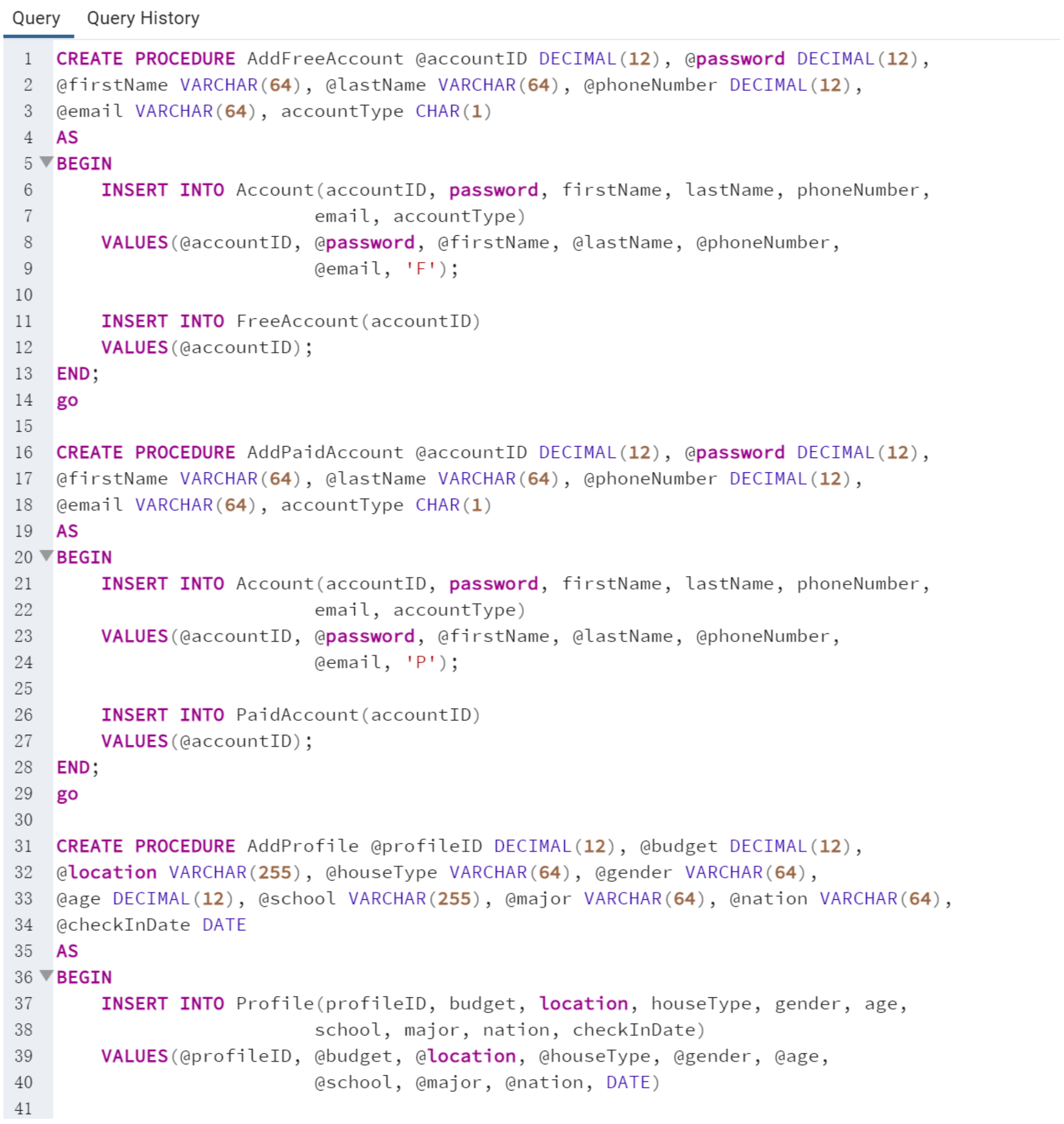
Lastly, it's reasonable that the school which the students study at will be a limiting column for some queries, such as queries that want to find roommates who studied at the same school so that they can go home together after classes . So I select the column school for an index, which would be a non-unique index because many Profiles and Requests have the same school.

Stored Procedure and Execution

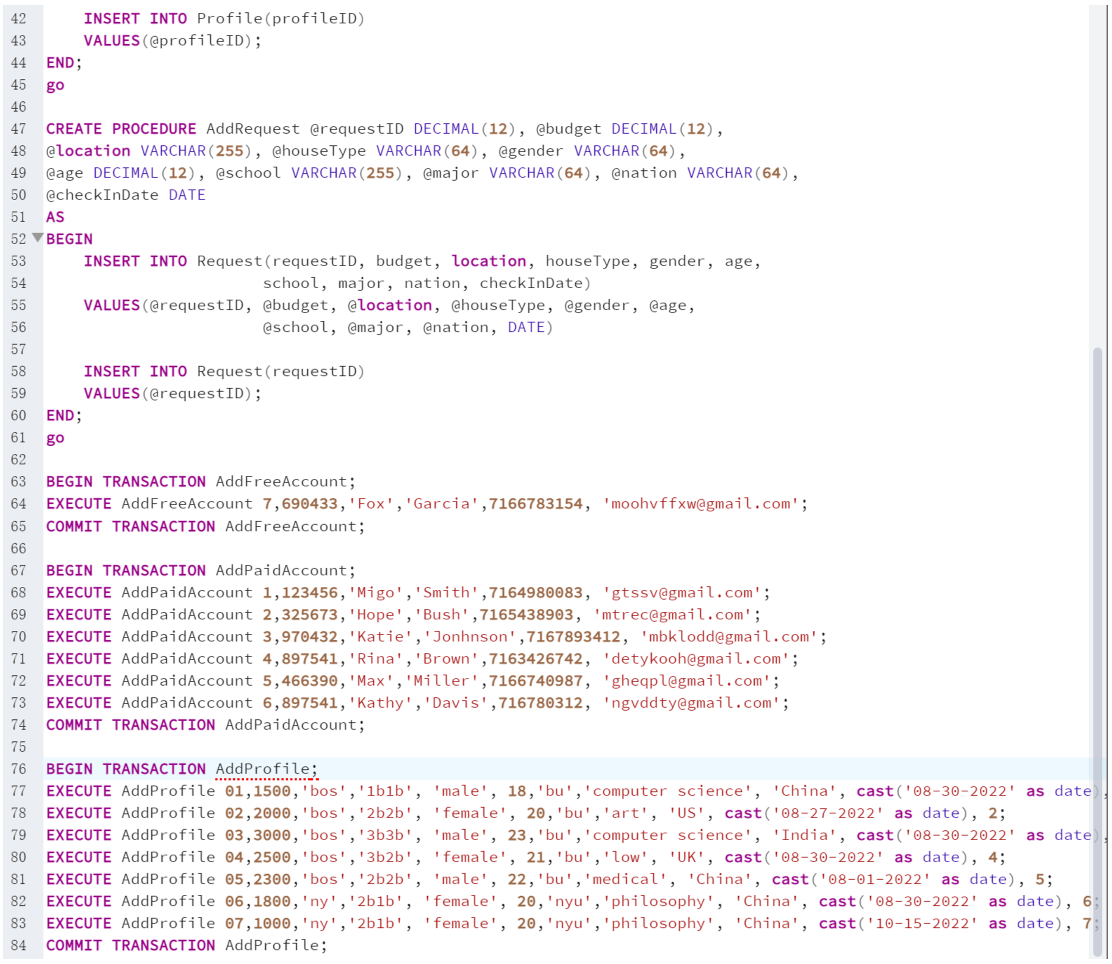
Our first use case is account signup, we have free account and paid account.

Our second use case is profile create.

Here is the screenshot of the stored procedure and execution.



```sql
CREATE PROCEDURE AddFreeAccount @accountID DECIMAL(12), @password DECIMAL(12),
@firstName VARCHAR(64), @lastName VARCHAR(64), @phoneNumber DECIMAL(12),
@email VARCHAR(64), accountType CHAR(1)
AS
BEGIN
    INSERT INTO Account(accountID, password, firstName, lastName, phoneNumber,
                       email, accountType)
    VALUES(@accountID, @password, @firstName, @lastName, @phoneNumber,
                       @email, 'F');

    INSERT INTO FreeAccount(accountID)
    VALUES(@accountID);
END;
go

CREATE PROCEDURE AddPaidAccount @accountID DECIMAL(12), @password DECIMAL(12),
@firstName VARCHAR(64), @lastName VARCHAR(64), @phoneNumber DECIMAL(12),
@email VARCHAR(64), accountType CHAR(1)
AS
BEGIN
    INSERT INTO Account(accountID, password, firstName, lastName, phoneNumber,
                       email, accountType)
    VALUES(@accountID, @password, @firstName, @lastName, @phoneNumber,
                       @email, 'P');

    INSERT INTO PaidAccount(accountID)
    VALUES(@accountID);
END;
go

CREATE PROCEDURE AddProfile @profileID DECIMAL(12), @budget DECIMAL(12),
@location VARCHAR(255), @houseType VARCHAR(64), @gender VARCHAR(64),
@age DECIMAL(12), @school VARCHAR(255), @major VARCHAR(64), @nation VARCHAR(64),
@checkInDate DATE
AS
BEGIN
    INSERT INTO Profile(profileID, budget, location, houseType, gender, age,
                       school, major, nation, checkInDate)
    VALUES(@profileID, @budget, @location, @houseType, @gender, @age,
                       @school, @major, @nation, DATE)

```

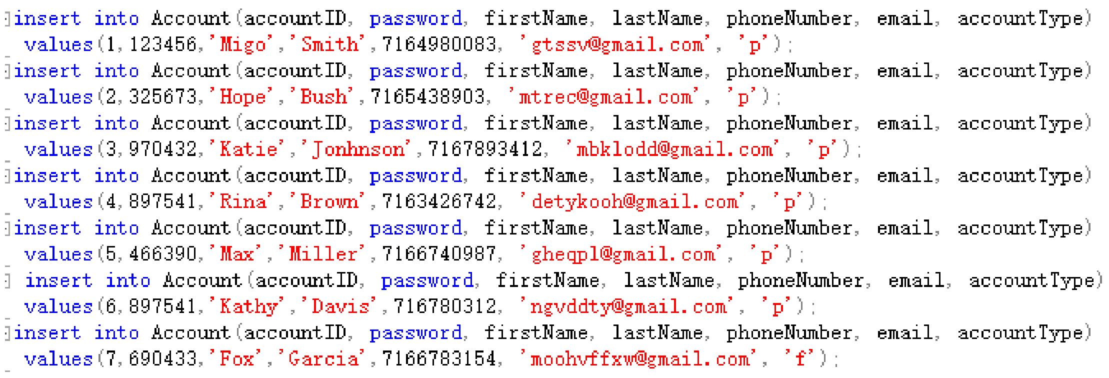


```sql
    INSERT INTO Profile(profileID)
    VALUES(@profileID);
END;
go

CREATE PROCEDURE AddRequest @requestID DECIMAL(12), @budget DECIMAL(12),
@location VARCHAR(255), @houseType VARCHAR(64), @gender VARCHAR(64),
@age DECIMAL(12), @school VARCHAR(255), @major VARCHAR(64), @nation VARCHAR(64),
@checkInDate DATE
AS
BEGIN
    INSERT INTO Request(requestID, budget, location, houseType, gender, age,
                       school, major, nation, checkInDate)
    VALUES(@requestID, @budget, @location, @houseType, @gender, @age,
                       @school, @major, @nation, DATE)

    INSERT INTO Request(requestID)
    VALUES(@requestID);
END;
go

BEGIN TRANSACTION AddFreeAccount;
EXECUTE AddFreeAccount 7,690433,'Fox','Garcia',7166783154, 'moohvffxw@gmail.com';
COMMIT TRANSACTION AddFreeAccount;

BEGIN TRANSACTION AddPaidAccount;
EXECUTE AddPaidAccount 1,123456,'Migo','Smith',7164980083, 'gtssv@gmail.com';
EXECUTE AddPaidAccount 2,325673,'Hope','Bush',7165438903, 'mtrec@gmail.com';
EXECUTE AddPaidAccount 3,970432,'Katie','Jonhnson',7167893412, 'mbklodd@gmail.com';
EXECUTE AddPaidAccount 4,897541,'Rina','Brown',7163426742, 'detykooh@gmail.com';
EXECUTE AddPaidAccount 5,466390,'Max','Miller',7166740987, 'gheqpl@gmail.com';
EXECUTE AddPaidAccount 6,897541,'Kathy','Davis',716780312, 'ngvddty@gmail.com';
COMMIT TRANSACTION AddPaidAccount;

BEGIN TRANSACTION AddProfile;
EXECUTE AddProfile 01,1500,'bos','1b1b', 'male', 18,'bu','computer science', 'China', cast('08-30-2022' as date)
EXECUTE AddProfile 02,2000,'bos','2b2b', 'female', 20,'bu','art', 'US', cast('08-27-2022' as date), 2;
EXECUTE AddProfile 03,3000,'bos','3b3b', 'male', 23,'bu','computer science', 'India', cast('08-30-2022' as date)
EXECUTE AddProfile 04,2500,'bos','3b2b', 'female', 21,'bu','low', 'UK', cast('08-30-2022' as date), 4;
EXECUTE AddProfile 05,2300,'bos','2b2b', 'male', 22,'bu','medical', 'China', cast('08-01-2022' as date), 5;
EXECUTE AddProfile 06,1800,'ny','2b1b', 'female', 20,'nyu','philosophy', 'China', cast('08-30-2022' as date), 6;
EXECUTE AddProfile 07,1000,'ny','2b1b', 'female', 20,'nyu','philosophy', 'China', cast('10-15-2022' as date), 7;
COMMIT TRANSACTION AddProfile;
```

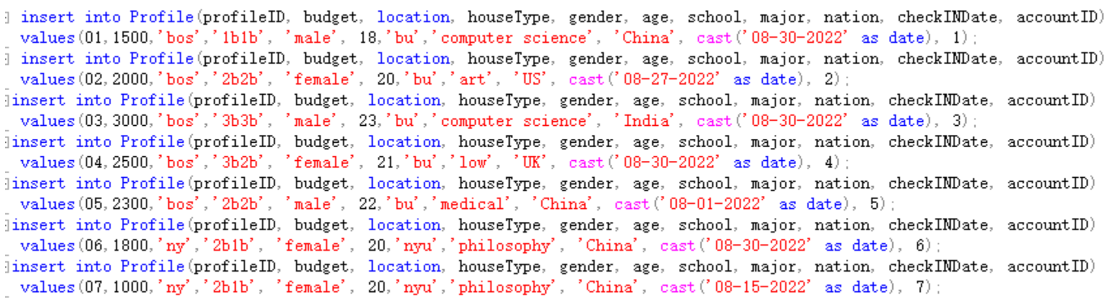
Under-Roof Buddy History

We use Match as a history. This is the attribute related to Match.

Attribute	**Description**
requestID	**This is the primary key of the history table. It is a DECIMAL(12) to allow for many values.**
matchDate	**This is the date the request occurred, with a DATE datatype**
matchedID	**This is the matched ID. It is a DECIMAL(12) to allow for many values.**
profileID	**This is the matched profileID. It is a DECIMAL(12) to allow for many values.**



```sql
insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(1,123456,'Migo','Smith',7164980083, 'gtssv@gmail.com', 'p');
insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(2,325673,'Hope','Bush',7165438903, 'mtrec@gmail.com', 'p');
insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(3,970432,'Katie','Jonhnson',7167893412, 'mbklodd@gmail.com', 'p');
insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(4,897541,'Rina','Brown',7163426742, 'detykooh@gmail.com', 'p');
insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(5,466390,'Max','Miller',7166740987, 'gheqpl@gmail.com', 'p');
 insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(6,897541,'Kathy','Davis',716780312, 'ngvddty@gmail.com', 'p');
insert into Account(accountID, password, firstName, lastName, phoneNumber, email, accountType)
  values(7,690433,'Fox','Garcia',7166783154, 'moohvffxw@gmail.com', 'f');
```

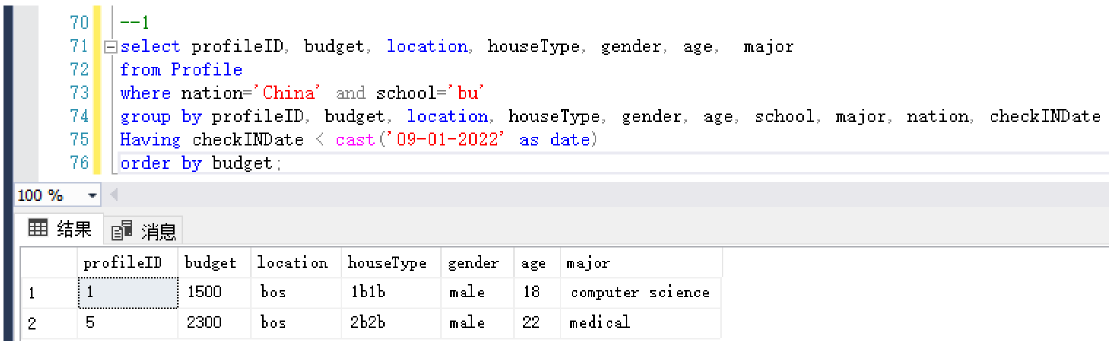


```sql
 insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(01,1500,'bos','1b1b', 'male', 18,'bu','computer science', 'China', cast('08-30-2022' as date), 1);
 insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(02,2000,'bos','2b2b', 'female', 20,'bu','art', 'US', cast('08-27-2022' as date), 2);
insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(03,3000,'bos','3b3b', 'male', 23,'bu','computer science', 'India', cast('08-30-2022' as date), 3);
insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(04,2500,'bos','3b2b', 'female', 21,'bu','low', 'UK', cast('08-30-2022' as date), 4);
insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(05,2300,'bos','2b2b', 'male', 22,'bu','medical', 'China', cast('08-01-2022' as date), 5);
insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(06,1800,'ny','2b1b', 'female', 20,'nyu','philosophy', 'China', cast('08-30-2022' as date), 6);
insert into Profile(profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID)
 values(07,1000,'ny','2b1b', 'female', 20,'nyu','philosophy', 'China', cast('08-15-2022' as date), 7);
```

Question1: What's the matching results of this request: the candidates' check-in dates are before 09/01/2022, their nation is China and their school is Boston University.

The reason why this query is important is that it represents one request of the user. Our database is to find the candidates who match the users. Through this query, the user can have the matching result from Profile, which represents the matching process. Less limits mean less requests.

Here is a screenshot of the query I use.



```sql
--1
select profileID, budget, location, houseType, gender, age,  major
from Profile
where nation='China' and school='bu'
group by profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate
Having checkINDate < cast('09-01-2022' as date)
order by budget;
```

100 %

结果 消息

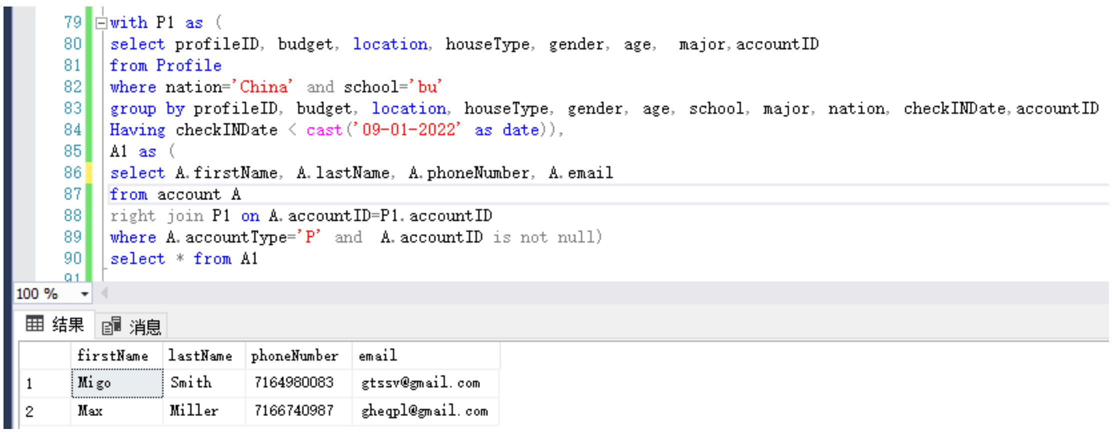
	profileID	budget	location	houseType	gender	age	major
1	1	1500	bos	1b1b	male	18	computer science
2	5	2300	bos	2b2b	male	22	medical

As shown above, to use the aggregate function to find the candidates whose check-in dates are before 09/01/2022, I first wrote the 'group by' sentence. I add some attributes to the 'group by' sentence so that I can see them in the resulting table. Then I wrote the 'where' sentence to find whose nation is China and whose school is Boston University, because I know some people may tend

to choose roommates who are from the same country and study at the same school so that they have more sharing topics. More importantly, international students would be inconvenienced if their check-in date is several days after they arrive in Boston, so I add this limit. Now I can see the candidates' renting budget to consider whether the budget I can afford, the place they want to reside to confirm we can live together, what house type they want to live in to think whether it is suitable for me, their gender which are the important considering factor, whether their age are closer to mine to share more topic, whether their major is similar to mine to study together.

Question2: After getting the matching results, I want to learn more about them， so I ask for their personal information to get in touch with them to decide whether they are really suitable to become my roommates.

Here is a screenshot of the query I use.



```
with P1 as (
select profileID, budget, location, houseType, gender, age,  major,accountID
from Profile
where nation='China' and school='bu'
group by profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate,accountID
Having checkINDate < cast('09-01-2022' as date)),
A1 as (
select A.firstName, A.lastName, A.phoneNumber, A.email
from account A
right join P1 on A.accountID=P1.accountID
where A.accountType='P' and  A.accountID is not null)
select * from A1
```

```
   firstName  lastName  phoneNumber  email
1  Migo       Smith     7164980083   gtssv@gmail.com
2  Max        Miller    7166740987   gheqpl@gmail.com
```

Firstly, I create my request as question1. After I got the result of these two matching cadidates, I named the resulting table as P1 so that I could continue using this table later. And then, I made the Account table join right to P1 table by the accountID so that I precisely got these two candidates' first name, last name, phoneNumber and email and then I can get in touch with them.

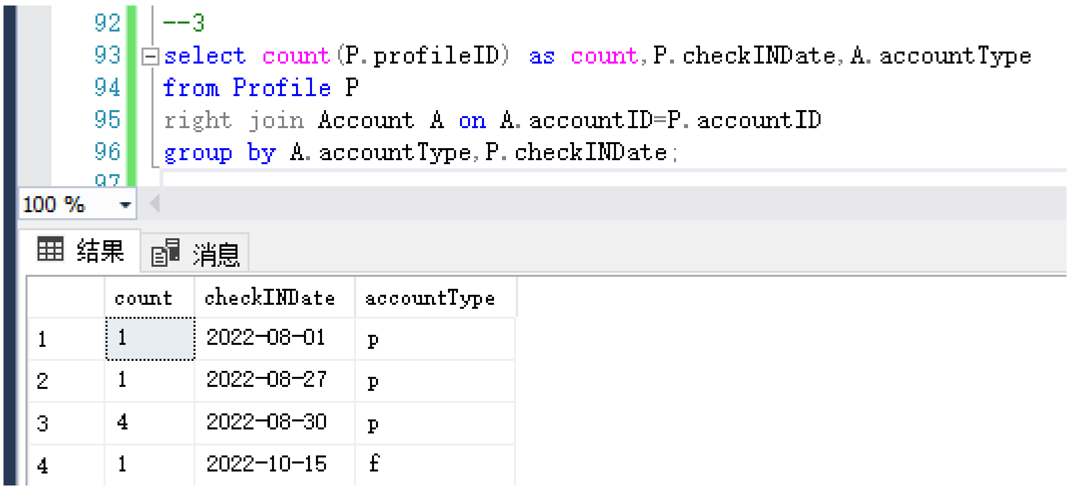
CODE	DESCRIPTION
with P1 as(query1), A1 as (query2)	The advantage of 'with' method is that it can write the code based on the normal order of human thinking. I first create P1, and then I create A1 on the basis of P1.
select profileID, budget, location, houseType, gender, age, major, accountID from Profile	This is the content that I want to get from the candidates.

where nation='China' and school='bu' **group by profileID, budget, location, houseType, gender, age, school, major, nation, checkINDate, accountID** **Having checkINDate < cast('09-01-2022' as date)**	**The resulting rows from the Profile Table are Chinese and Boston University's student.** **After grouping them by these attributes, I can filter the candidates whose check-in dates are before 09/01/2022, so that I can obtain only the desired rows.**
select A.firstName, A.lastName, A.phoneNumber, A.email **from account A**	**I select the personal information from the candidates attribute**
right join P1 on A.accountID=P1.accountID	**The reason why I write right join rather than left join is that I only want to take the personal information** **of candidates who are in the filtering result table P1. If I use the left join, it will show all the users information in the Account Table.**
where A.accountType='P' **and A.accountID is not null**	**Only the Pain Account can take part in getting the personal information of matching candidates.**
select * from A1	**A1 Table is the result of Account Table that joins the right P1 Table. So get all the information that is the final filtering results.**

Question3: What factor influences the amount of paid accounts?

This question is important for our promoters, we want to have more and more users and help a larger number of users to find suitable roommates. So researching the reasons for people to pay the monthly fee is useful. Here, I want to study the relationship between check-in dates and paid accounts. Suppose that now it's July, 2022. I found that getting closer to check-in dates, it was more likely for users to pay the monthly account fee so that they could get the future roommates contact information. This query can help us to focus on promoting that we can help users to find roommates effectively to gain a larger market.

Here is a screenshot of the query I use.


```
--3
select count(P.profileID) as count,P.checkINDate,A.accountType
from Profile P
right join Account A on A.accountID=P.accountID
group by A.accountType,P.checkINDate;
```

```
   count  checkINDate  accountType
1  1      2022-08-01   p
2  1      2022-08-27   p
3  4      2022-08-30   p
4  1      2022-10-15   f
```

CODE	DESCRIPTION
group by A.accountType,P.checkINDate	**I divide the Account Table into Free and Paid accounts as well as check-in dates so that I can get the results showing the check-in dates.**
select count(P.profileID) as count,P.checkINDate,A.accountType from Profile P	**Obtain the data about Free and Paid accounts, check-in dates and count the number of users in each group.**
right join Account A on A.accountID=P.accountID	**Because the accountType is not in the Profile Table, I joined it with the Account Table that has the accountType.**

Summary and Reflection

Our database is a website/app named Under-Roof Buddy, which helps many international students find their roommates. It focuses on collecting the personal information of registered users and then forming the users' own data for others to select.

At the same time, users are asked to provide information such as housing requirements and expectations for roommates, which filters out roommates who meet the requirements.

Furthermore, Under-Roof Buddy provides more personal information such as the contact information of roommates, allowing users to communicate more deeply with their candidates.

The structural database rules and ERD for my database design contain the important entities of Request, Profile, and Account, as well as relationships between them.

Finally, We also need to find out how to match the requests of users properly in the database.

Under-Roof Buddy wants to be a porter of information and is committed to help international students adjust to the foreign environment more pleasantly.