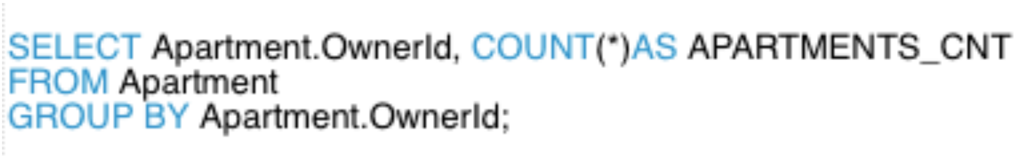
PM3_Pioneers OCT_25_2018

Created by: Qian Wang, Nan Liu, Shufan Xing, Wen Zhang

Part1: 10 related query questions:

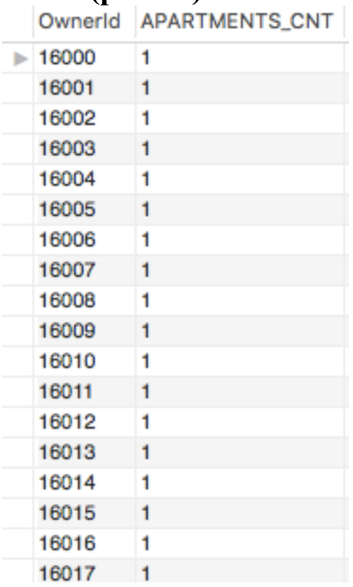
#1. How many apartments are owned by a given landlord?

Code:



```sql
SELECT Apartment.OwnerId, COUNT(*)AS APARTMENTS_CNT
FROM Apartment
GROUP BY Apartment.OwnerId;
```

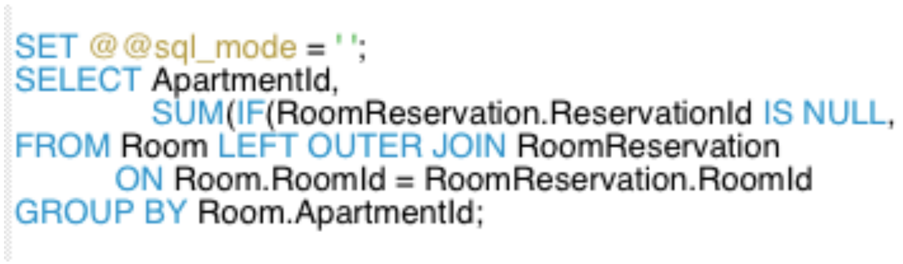
Result(partial):



OwnerId	APARTMENTS_CNT
16000	1
16001	1
16002	1
16003	1
16004	1
16005	1
16006	1
16007	1
16008	1
16009	1
16010	1
16011	1
16012	1
16013	1
16014	1
16015	1
16016	1
16017	1

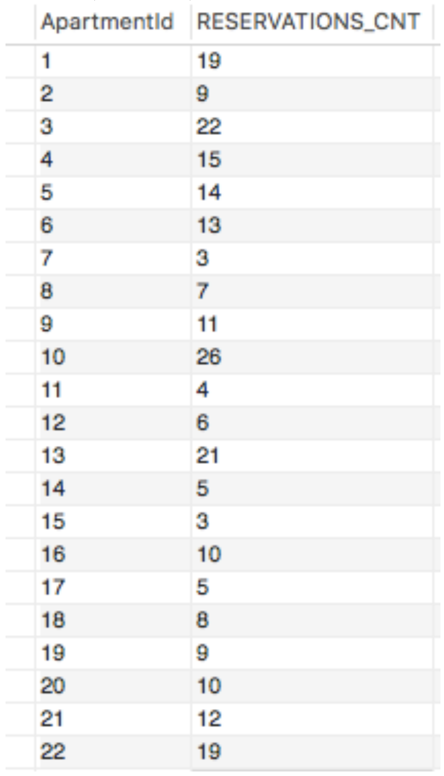
#2. How many reservations are made on a given apartment?

Code:



```
SET @@sql_mode = ' ';
SELECT ApartmentId,
            SUM(IF(RoomReservation.ReservationId IS NULL,
FROM Room LEFT OUTER JOIN RoomReservation
         ON Room.RoomId = RoomReservation.RoomId
GROUP BY Room.ApartmentId;
```

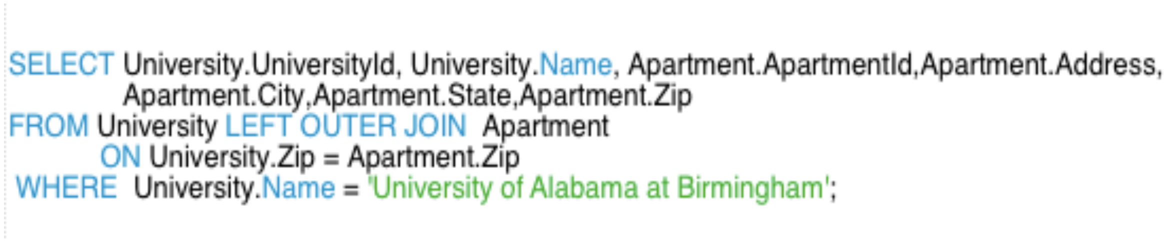
Result(Partial):



ApartmentId	RESERVATIONS_CNT
1	19
2	9
3	22
4	15
5	14
6	13
7	3
8	7
9	11
10	26
11	4
12	6
13	21
14	5
15	3
16	10
17	5
18	8
19	9
20	10
21	12
22	19

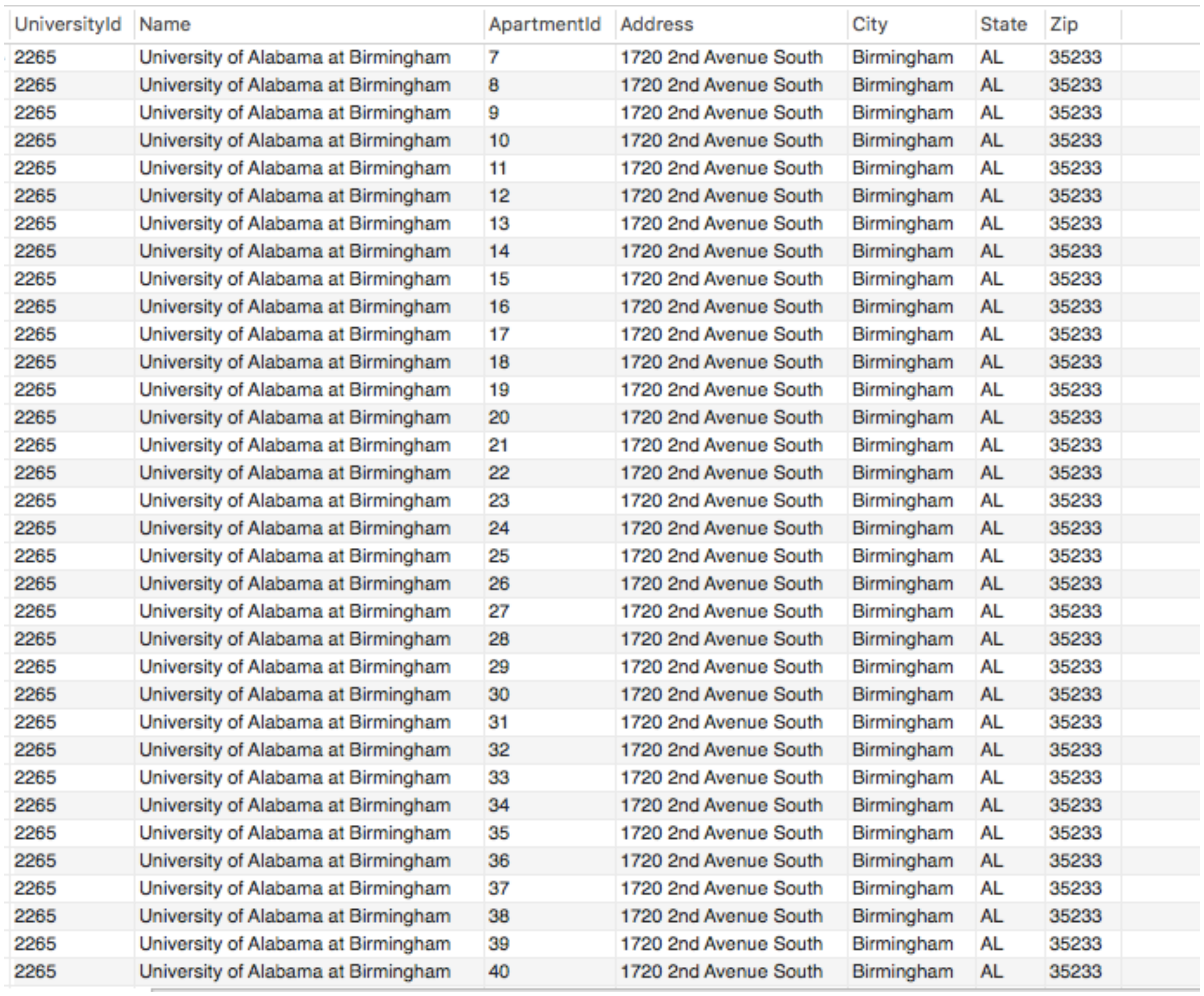
#3. What are the available apartments near given University (University Name = 'University of Alabama at Birmingham')?
Search by University Name and return apartments who has same zip code.
List out apartment ID and address.

Code:



```
SELECT University.UniversityId, University.Name, Apartment.ApartmentId,Apartment.Address,
          Apartment.City,Apartment.State,Apartment.Zip
FROM University LEFT OUTER JOIN  Apartment
        ON University.Zip = Apartment.Zip
 WHERE  University.Name = 'University of Alabama at Birmingham';
```

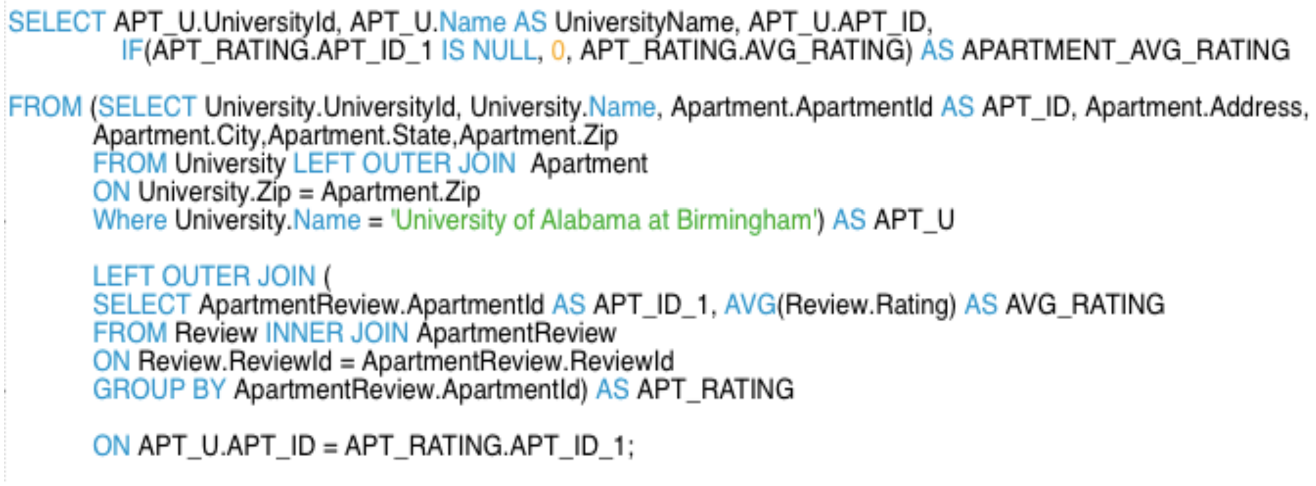
Result(partial):



UniversityId	Name	ApartmentId	Address	City	State	Zip
2265	University of Alabama at Birmingham	7	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	8	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	9	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	10	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	11	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	12	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	13	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	14	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	15	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	16	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	17	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	18	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	19	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	20	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	21	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	22	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	23	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	24	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	25	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	26	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	27	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	28	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	29	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	30	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	31	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	32	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	33	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	34	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	35	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	36	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	37	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	38	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	39	1720 2nd Avenue South	Birmingham	AL	35233
2265	University of Alabama at Birmingham	40	1720 2nd Avenue South	Birmingham	AL	35233

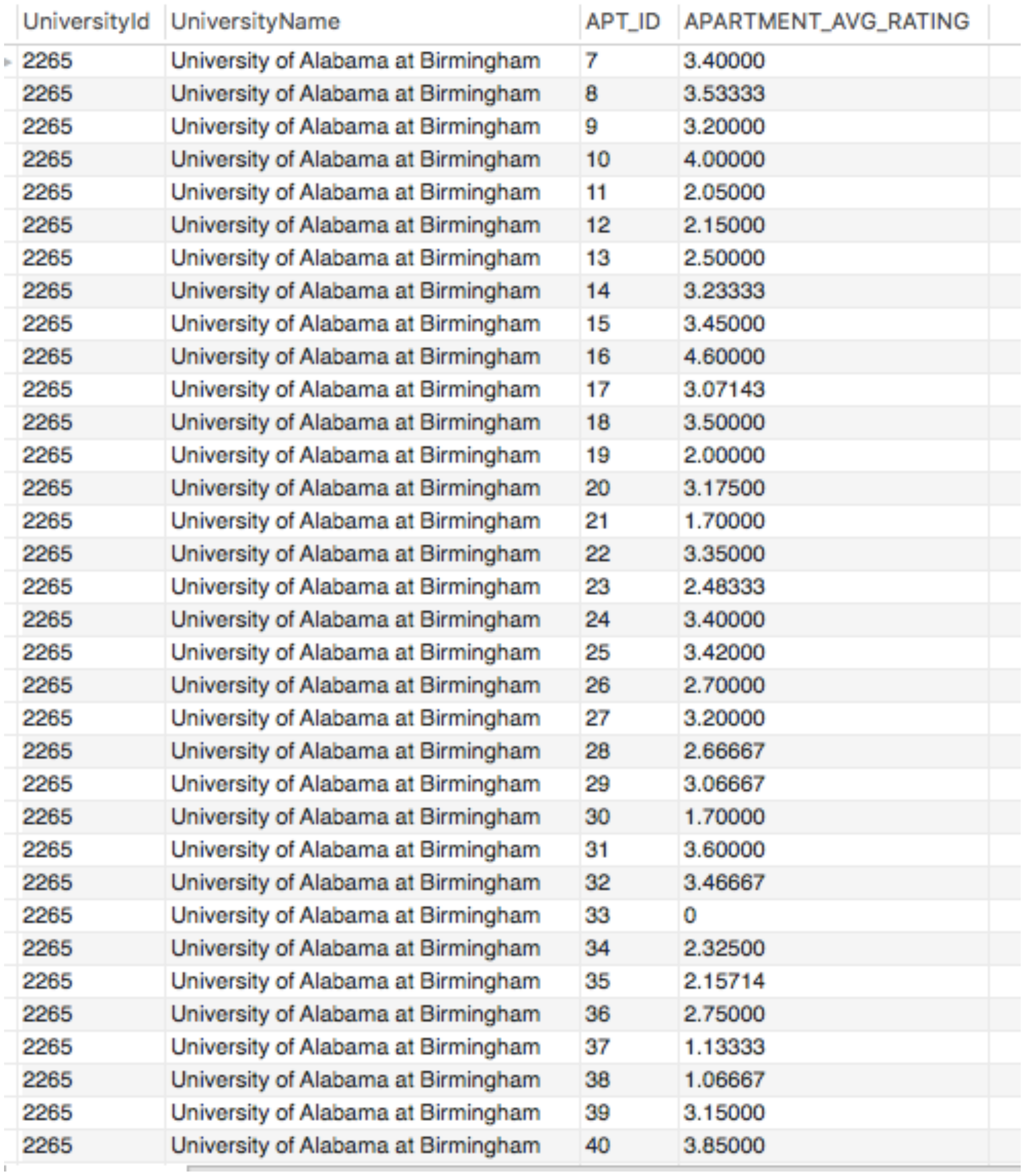
#4. What are the average review rating for each available apartment around one certain University (UniversityName = 'University of Alabama at Birmingham')?
Code:

For read



```
SELECT APT_U.UniversityId, APT_U.Name AS UniversityName, APT_U.APT_ID,
        IF(APT_RATING.APT_ID_1 IS NULL, 0, APT_RATING.AVG_RATING) AS APARTMENT_AVG_RATING

FROM (SELECT University.UniversityId, University.Name, Apartment.ApartmentId AS APT_ID, Apartment.Address,
       Apartment.City,Apartment.State,Apartment.Zip
       FROM University LEFT OUTER JOIN  Apartment
       ON University.Zip = Apartment.Zip
       Where University.Name = 'University of Alabama at Birmingham') AS APT_U

       LEFT OUTER JOIN (
       SELECT ApartmentReview.ApartmentId AS APT_ID_1, AVG(Review.Rating) AS AVG_RATING
       FROM Review INNER JOIN ApartmentReview
       ON Review.ReviewId = ApartmentReview.ReviewId
       GROUP BY ApartmentReview.ApartmentId) AS APT_RATING

       ON APT_U.APT_ID = APT_RATING.APT_ID_1;
```

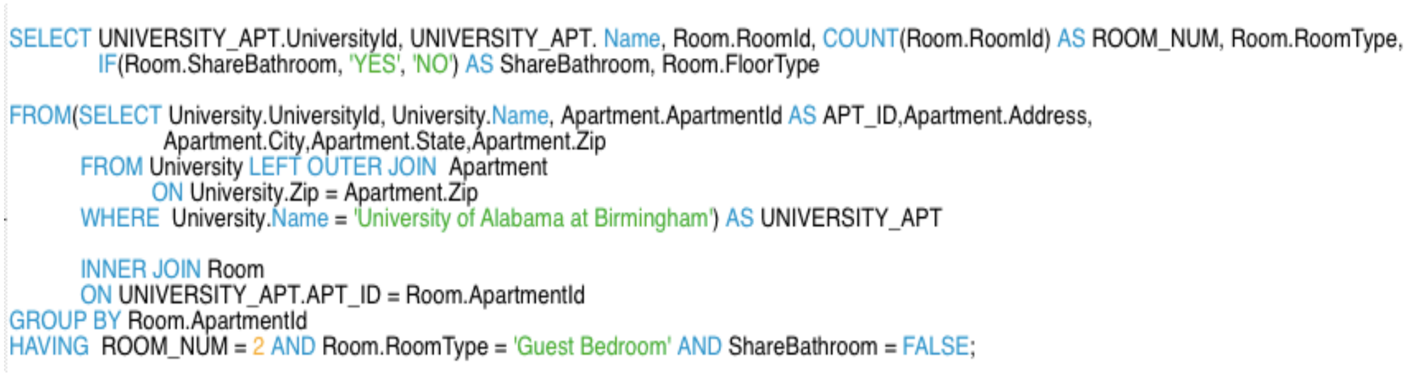
Results(Partial):



UniversityId	UniversityName	APT_ID	APARTMENT_AVG_RATING
2265	University of Alabama at Birmingham	7	3.40000
2265	University of Alabama at Birmingham	8	3.53333
2265	University of Alabama at Birmingham	9	3.20000
2265	University of Alabama at Birmingham	10	4.00000
2265	University of Alabama at Birmingham	11	2.05000
2265	University of Alabama at Birmingham	12	2.15000
2265	University of Alabama at Birmingham	13	2.50000
2265	University of Alabama at Birmingham	14	3.23333
2265	University of Alabama at Birmingham	15	3.45000
2265	University of Alabama at Birmingham	16	4.60000
2265	University of Alabama at Birmingham	17	3.07143
2265	University of Alabama at Birmingham	18	3.50000
2265	University of Alabama at Birmingham	19	2.00000
2265	University of Alabama at Birmingham	20	3.17500
2265	University of Alabama at Birmingham	21	1.70000
2265	University of Alabama at Birmingham	22	3.35000
2265	University of Alabama at Birmingham	23	2.48333
2265	University of Alabama at Birmingham	24	3.40000
2265	University of Alabama at Birmingham	25	3.42000
2265	University of Alabama at Birmingham	26	2.70000
2265	University of Alabama at Birmingham	27	3.20000
2265	University of Alabama at Birmingham	28	2.66667
2265	University of Alabama at Birmingham	29	3.06667
2265	University of Alabama at Birmingham	30	1.70000
2265	University of Alabama at Birmingham	31	3.60000
2265	University of Alabama at Birmingham	32	3.46667
2265	University of Alabama at Birmingham	33	0
2265	University of Alabama at Birmingham	34	2.32500
2265	University of Alabama at Birmingham	35	2.15714
2265	University of Alabama at Birmingham	36	2.75000
2265	University of Alabama at Birmingham	37	1.13333
2265	University of Alabama at Birmingham	38	1.06667
2265	University of Alabama at Birmingham	39	3.15000
2265	University of Alabama at Birmingham	40	3.85000

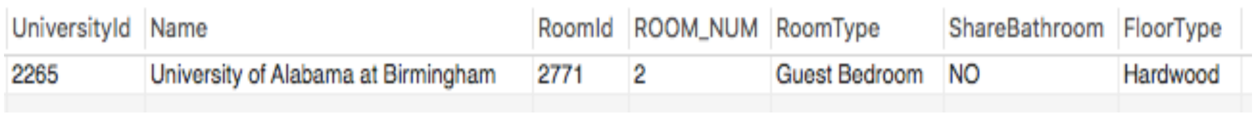
#5. What are the available rooms that satisfied user's specific requirement?
Given a university name (University of Alabama at Birmingham) find all available apartments nearby and then filter available rooms by room number = 2, room type = GuestBedroom, ShareBathroom = NO

Code:
For read:



```
SELECT UNIVERSITY_APT.UniversityId, UNIVERSITY_APT. Name, Room.RoomId, COUNT(Room.RoomId) AS ROOM_NUM, Room.RoomType,
        IF(Room.ShareBathroom, 'YES', 'NO') AS ShareBathroom, Room.FloorType

FROM(SELECT University.UniversityId, University.Name, Apartment.ApartmentId AS APT_ID,Apartment.Address,
              Apartment.City,Apartment.State,Apartment.Zip
      FROM University LEFT OUTER JOIN  Apartment
            ON University.Zip = Apartment.Zip
      WHERE  University.Name = 'University of Alabama at Birmingham') AS UNIVERSITY_APT

      INNER JOIN Room
      ON UNIVERSITY_APT.APT_ID = Room.ApartmentId
GROUP BY Room.ApartmentId
HAVING  ROOM_NUM = 2 AND Room.RoomType = 'Guest Bedroom' AND ShareBathroom = FALSE;
```

Result:

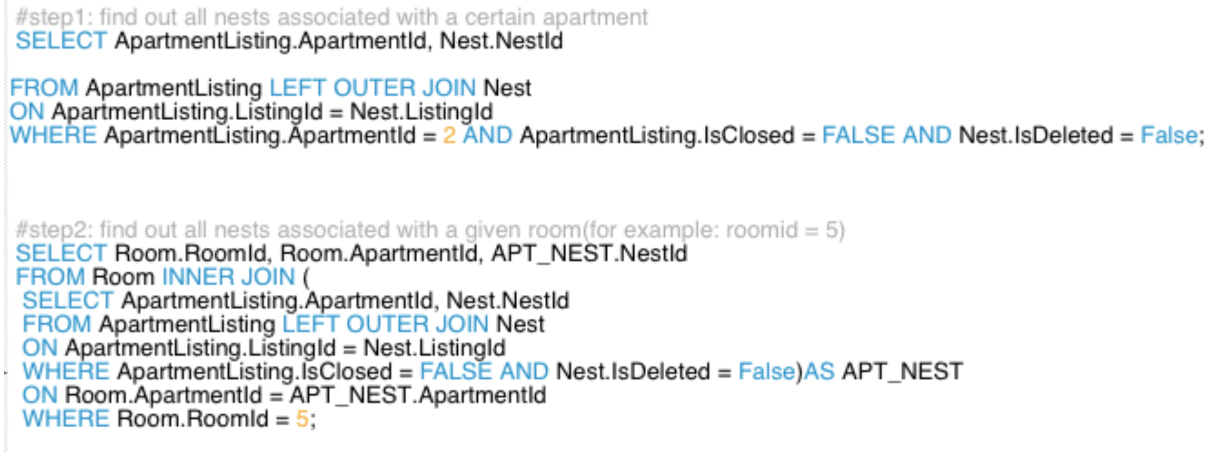


UniversityId	Name	RoomId	ROOM_NUM	RoomType	ShareBathroom	FloorType
2265	University of Alabama at Birmingham	2771	2	Guest Bedroom	NO	Hardwood

#6. How many nests are associated with a given room（room id = 5）? List all the nest ID.

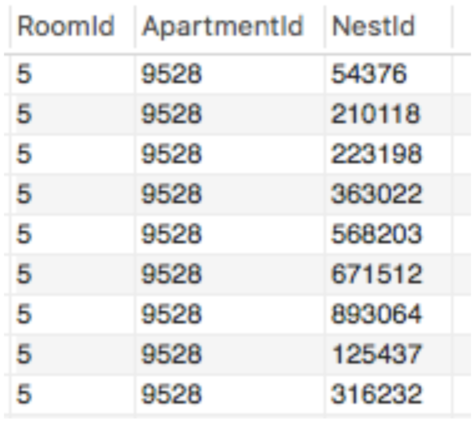
Code:

For read:



```
#step1: find out all nests associated with a certain apartment
SELECT ApartmentListing.ApartmentId, Nest.NestId

FROM ApartmentListing LEFT OUTER JOIN Nest
ON ApartmentListing.ListingId = Nest.ListingId
WHERE ApartmentListing.ApartmentId = 2 AND ApartmentListing.IsClosed = FALSE AND Nest.IsDeleted = False;


#step2: find out all nests associated with a given room(for example: roomid = 5)
SELECT Room.RoomId, Room.ApartmentId, APT_NEST.NestId
FROM Room INNER JOIN (
 SELECT ApartmentListing.ApartmentId, Nest.NestId
 FROM ApartmentListing LEFT OUTER JOIN Nest
 ON ApartmentListing.ListingId = Nest.ListingId
 WHERE ApartmentListing.IsClosed = FALSE AND Nest.IsDeleted = False)AS APT_NEST
 ON Room.ApartmentId = APT_NEST.ApartmentId
 WHERE Room.RoomId = 5;
```

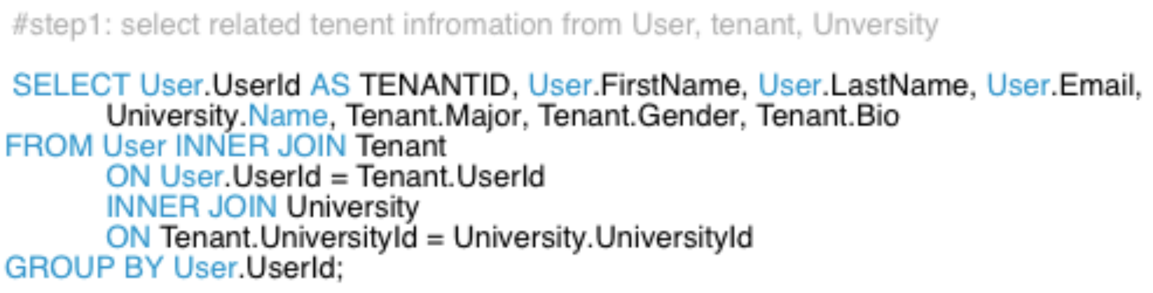
Result:



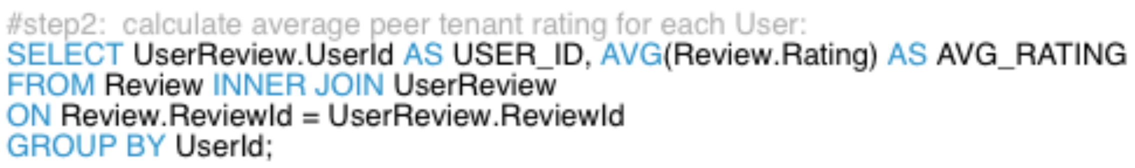
RoomId	ApartmentId	NestId
5	9528	54376
5	9528	210118
5	9528	223198
5	9528	363022
5	9528	568203
5	9528	671512
5	9528	893064
5	9528	125437
5	9528	316232

#7. List information of all the Tenants in a certain nest (eg: nestId = 20), like first name, last name, university name, major, gender, Bio, average peer tenant review rating.

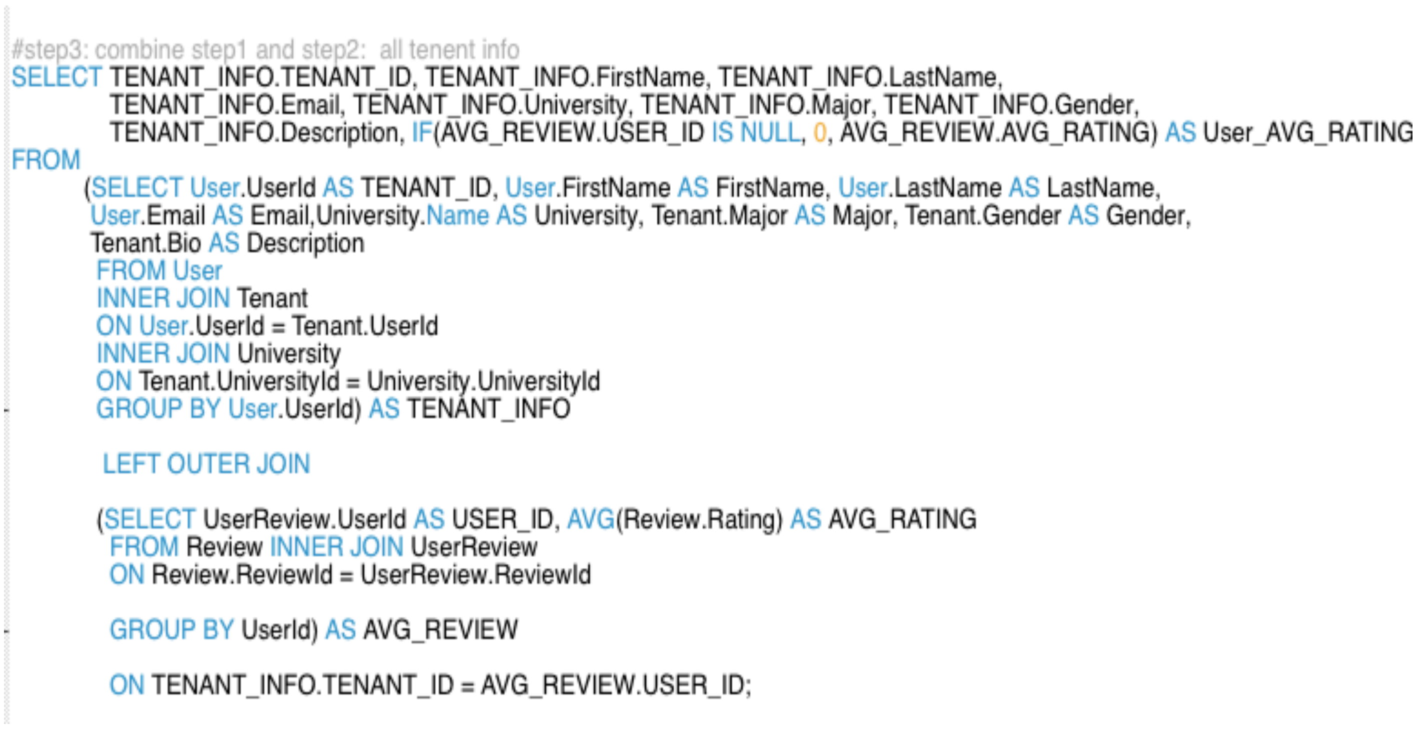
Code:



```
 #step1: select related tenent infromation from User, tenant, Unversity

 SELECT User.UserId AS TENANTID, User.FirstName, User.LastName, User.Email,
        University.Name, Tenant.Major, Tenant.Gender, Tenant.Bio
FROM User INNER JOIN Tenant
        ON User.UserId = Tenant.UserId
        INNER JOIN University
        ON Tenant.UniversityId = University.UniversityId
GROUP BY User.UserId;
```



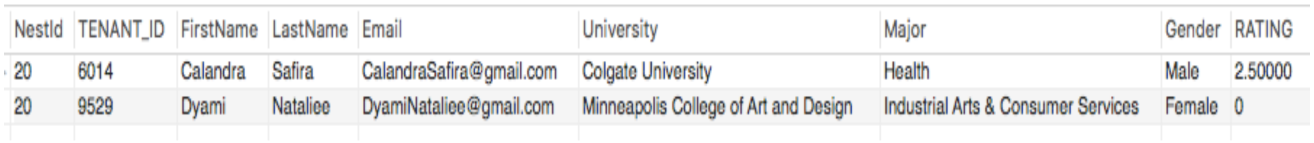
```
#step2:  calculate average peer tenant rating for each User:
SELECT UserReview.UserId AS USER_ID, AVG(Review.Rating) AS AVG_RATING
FROM Review INNER JOIN UserReview
ON Review.ReviewId = UserReview.ReviewId
GROUP BY UserId;
```



```
#step3: combine step1 and step2:  all tenent info
SELECT TENANT_INFO.TENANT_ID, TENANT_INFO.FirstName, TENANT_INFO.LastName,
        TENANT_INFO.Email, TENANT_INFO.University, TENANT_INFO.Major, TENANT_INFO.Gender,
        TENANT_INFO.Description, IF(AVG_REVIEW.USER_ID IS NULL, 0, AVG_REVIEW.AVG_RATING) AS User_AVG_RATING
FROM
      (SELECT User.UserId AS TENANT_ID, User.FirstName AS FirstName, User.LastName AS LastName,
       User.Email AS Email,University.Name AS University, Tenant.Major AS Major, Tenant.Gender AS Gender,
       Tenant.Bio AS Description
        FROM User
        INNER JOIN Tenant
        ON User.UserId = Tenant.UserId
        INNER JOIN University
        ON Tenant.UniversityId = University.UniversityId
        GROUP BY User.UserId) AS TENANT_INFO

         LEFT OUTER JOIN

        (SELECT UserReview.UserId AS USER_ID, AVG(Review.Rating) AS AVG_RATING
         FROM Review INNER JOIN UserReview
         ON Review.ReviewId = UserReview.ReviewId

         GROUP BY UserId) AS AVG_REVIEW

         ON TENANT_INFO.TENANT_ID = AVG_REVIEW.USER_ID;
```

```
#step4: Tenents in nest (nestId = 20)
SELECT Nest.NestId, RoomReservation.TenantId
FROM Nest INNER JOIN RoomReservation
ON Nest.NestId = RoomReservation.NestId
WHERE Nest.NestId = 20;


#step 5: combine #3 and #4

SELECT Nest.NestId, TENANT_INFO.ID AS TENANT_ID, TENANT_INFO.FirstName, TENANT_INFO.LastName,
        TENANT_INFO.Email, TENANT_INFO.University, TENANT_INFO.Major,
        TENANT_INFO.Gender,
        IF(AVG_REVIEW.ID_TWO IS NULL, 0, AVG_REVIEW.AVG_RATING) AS RATING

FROM Nest INNER JOIN RoomReservation
       ON Nest.NestId = RoomReservation.NestId
       INNER JOIN

       (SELECT User.UserId AS ID, User.FirstName AS FirstName, User.LastName AS LastName,
                User.Email AS Email,University.Name AS University, Tenant.Major AS Major, Tenant.Gender AS Gender,
                Tenant.Bio AS Description
        FROM User
              INNER JOIN Tenant
              ON User.UserId = Tenant.UserId
              INNER JOIN University
              ON Tenant.UniversityId = University.UniversityId
       GROUP BY User.UserId) AS TENANT_INFO

       ON RoomReservation.TenantId = TENANT_INFO.ID

        LEFT OUTER JOIN

        (SELECT UserReview.UserId AS ID_TWO, AVG(Review.Rating) AS AVG_RATING
         FROM Review INNER JOIN UserReview
         ON Review.ReviewId = UserReview.ReviewId
         #WHERE UserReview.Type = 'Tenant'
         GROUP BY UserId) AS AVG_REVIEW

         ON TENANT_INFO.ID = AVG_REVIEW.ID_TWO

WHERE Nest.NestId = 20;
```

Result:



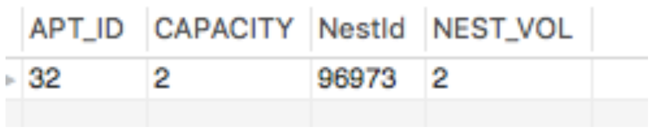
NestId	TENANT_ID	FirstName	LastName	Email	University	Major	Gender	RATING
20	6014	Calandra	Safira	CalandraSafira@gmail.com	Colgate University	Health	Male	2.50000
20	9529	Dyami	Nataliee	DyamiNataliee@gmail.com	Minneapolis College of Art and Design	Industrial Arts & Consumer Services	Female	0

**#8. How many nests have reached the Apartment capacity (for Apartment Id = 26)?
for certain apartment (current listing), lists all nests that reach the apartment capacity.**

Code:

```
#step 1: calculate reservations per nest
 SELECT Nest.NestId , RoomReservation.ReservationId,COUNT(RoomReservation.ReservationId) AS RESERVATION_NUM_PER_NEST
 FROM Nest INNER JOIN RoomReservation
 ON RoomReservation.NestId = Nest.NestId
 GROUP BY Nest.NestId;

# step2: find out nests for Apartment26 current listing
SELECT ApartmentListing.ListingId AS LIST_ID, ApartmentListing.ApartmentId AS APT_ID_1,
       NEST_INFO.NestId, NEST_INFO.NEST_VOL
FROM ApartmentListing

      LEFT OUTER JOIN

      (SELECT Nest.NestId, Nest.ListingId AS LIST_ID_1, COUNT(RoomReservation.ReservationId) AS NEST_VOL
       FROM Nest INNER JOIN RoomReservation
       ON RoomReservation.NestId = Nest.NestId
       WHERE  Nest.IsDeleted = FALSE
       GROUP BY Nest.NestId) AS NEST_INFO

       ON ApartmentListing.ListingId = NEST_INFO.LIST_ID_1

WHERE ApartmentListing.IsClosed = FALSE AND ApartmentListing.ApartmentId = 32;


#step3: find out apartment capacity
SELECT Apartment.ApartmentId AS APT_ID, Floorplan.NumberOfBedrooms AS CAPACITY

FROM  Apartment INNER JOIN Floorplan
      ON Apartment.FloorPlanId = Floorplan.FloorPlanId

GROUP BY ApartmentId;

#step4:  for each apartment(current apartmentlist), lists all nests that reach the apartment capactiy


SELECT APT_CAP.APT_ID, APT_CAP.CAPACITY, LIST_NEST.NestId, LIST_NEST.NEST_VOL

FROM (SELECT Apartment.ApartmentId AS APT_ID, Floorplan.NumberOfBedrooms AS CAPACITY
      FROM  Apartment INNER JOIN Floorplan
            ON Apartment.FloorPlanId = Floorplan.FloorPlanId
      GROUP BY ApartmentId) AS APT_CAP

      INNER JOIN

      (SELECT ApartmentListing.ListingId AS LIST_ID, ApartmentListing.ApartmentId AS APT_ID_1,
              NEST_INFO.NestId, NEST_INFO.NEST_VOL
       FROM  ApartmentListing
             LEFT OUTER JOIN
             (SELECT Nest.NestId, Nest.ListingId AS LIST_ID_1, COUNT(RoomReservation.ReservationId) AS NEST_VOL
              FROM Nest INNER JOIN RoomReservation
              ON RoomReservation.NestId = Nest.NestId
              WHERE  Nest.IsDeleted = FALSE
              GROUP BY Nest.NestId) AS NEST_INFO
             ON ApartmentListing.ListingId = NEST_INFO.LIST_ID_1
      WHERE ApartmentListing.IsClosed = FALSE AND ApartmentListing.ApartmentId = 32 ) AS LIST_NEST


      ON APT_CAP.APT_ID = LIST_NEST.APT_ID_1

HAVING LIST_NEST.NEST_VOL = APT_CAP.CAPACITY;
```

Result:



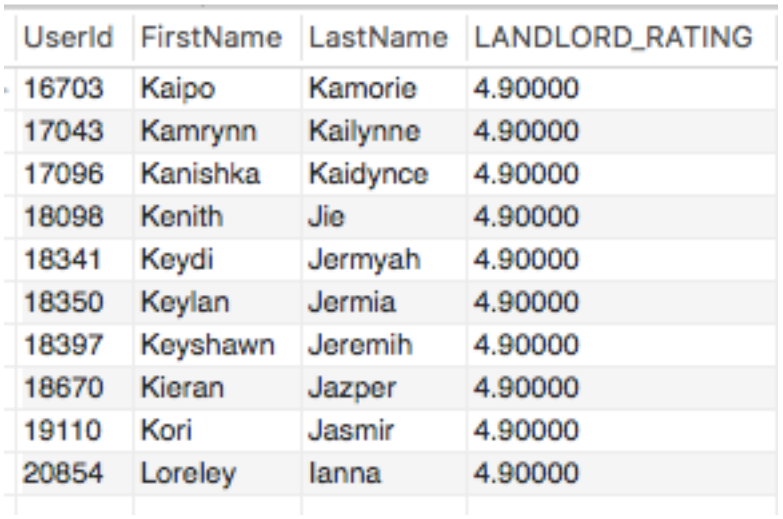
APT_ID	CAPACITY	NestId	NEST_VOL
32	2	96973	2

#9. Who are the top 10 highest rating land lord this year?

Code:

```
SELECT LAND_LORD.UserId, LAND_LORD.FirstName, LAND_LORD.LastName,
         IF(LANDLORD_REVIEW.UserId IS NULL, 0, AVG_RATING) AS LANDLORD_RATING

FROM (SELECT User.UserId, User.FirstName, User.LastName
        FROM User INNER JOIN Landlord
              ON User.UserId = Landlord.UserId) AS LAND_LORD

       LEFT OUTER JOIN

       (SELECT UserReview.UserId, AVG(Review.Rating) AS AVG_RATING
         FROM Review INNER JOIN UserReview
              ON Review.ReviewId = UserReview.ReviewId
         #WHERE UserReview.Type = 'Landlord'
         GROUP BY UserReview.UserId) AS LANDLORD_REVIEW

       ON LAND_LORD.UserId = LANDLORD_REVIEW.UserId

GROUP BY LAND_LORD.UserId

ORDER BY LANDLORD_RATING DESC, LAND_LORD.FirstName ASC, LAND_LORD.LastName ASC

LIMIT 10;
```

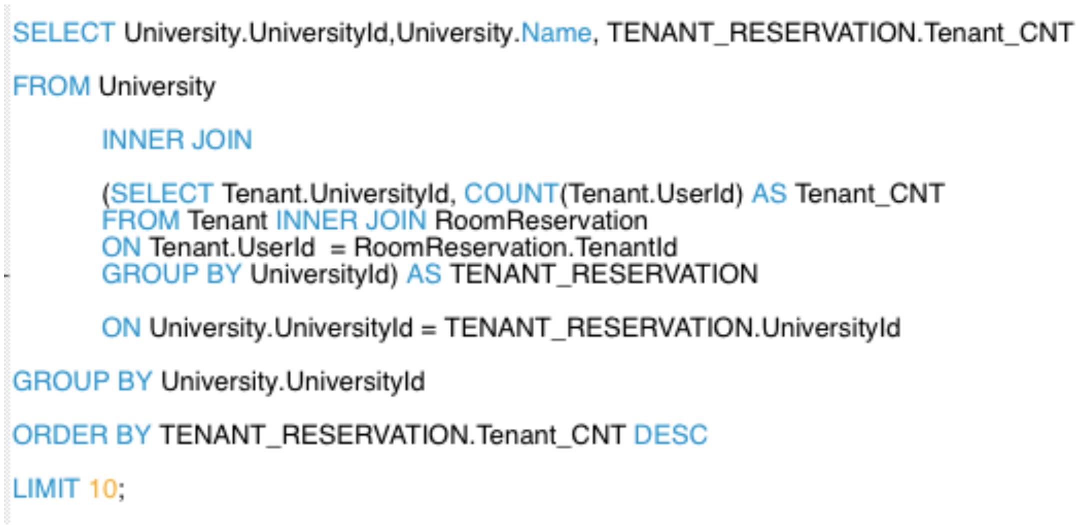
Result:



UserId	FirstName	LastName	LANDLORD_RATING
16703	Kaipo	Kamorie	4.90000
17043	Kamrynn	Kailynne	4.90000
17096	Kanishka	Kaidynce	4.90000
18098	Kenith	Jie	4.90000
18341	Keydi	Jermyah	4.90000
18350	Keylan	Jermia	4.90000
18397	Keyshawn	Jeremih	4.90000
18670	Kieran	Jazper	4.90000
19110	Kori	Jasmir	4.90000
20854	Loreley	Ianna	4.90000

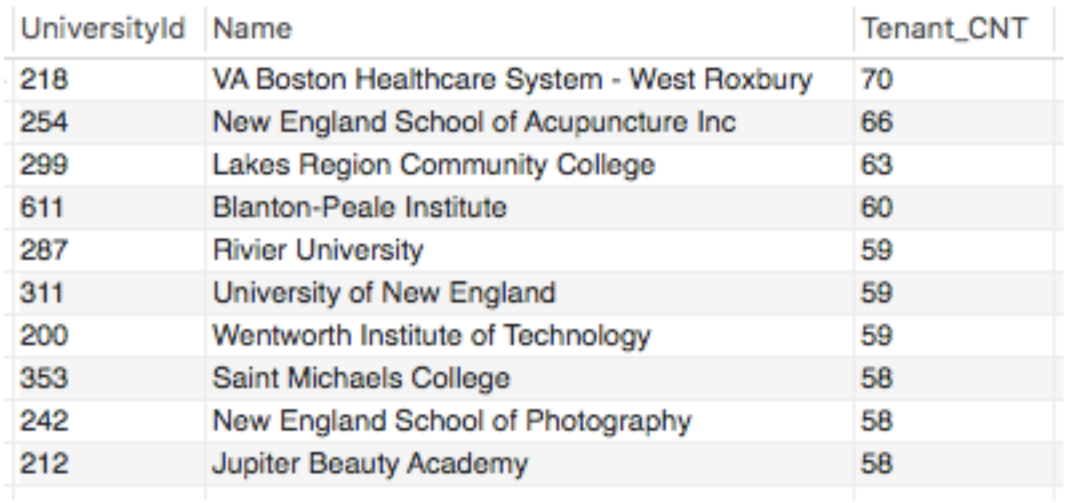
#10. What are the top 10 universities that has maximum housing demand this year?

Code:



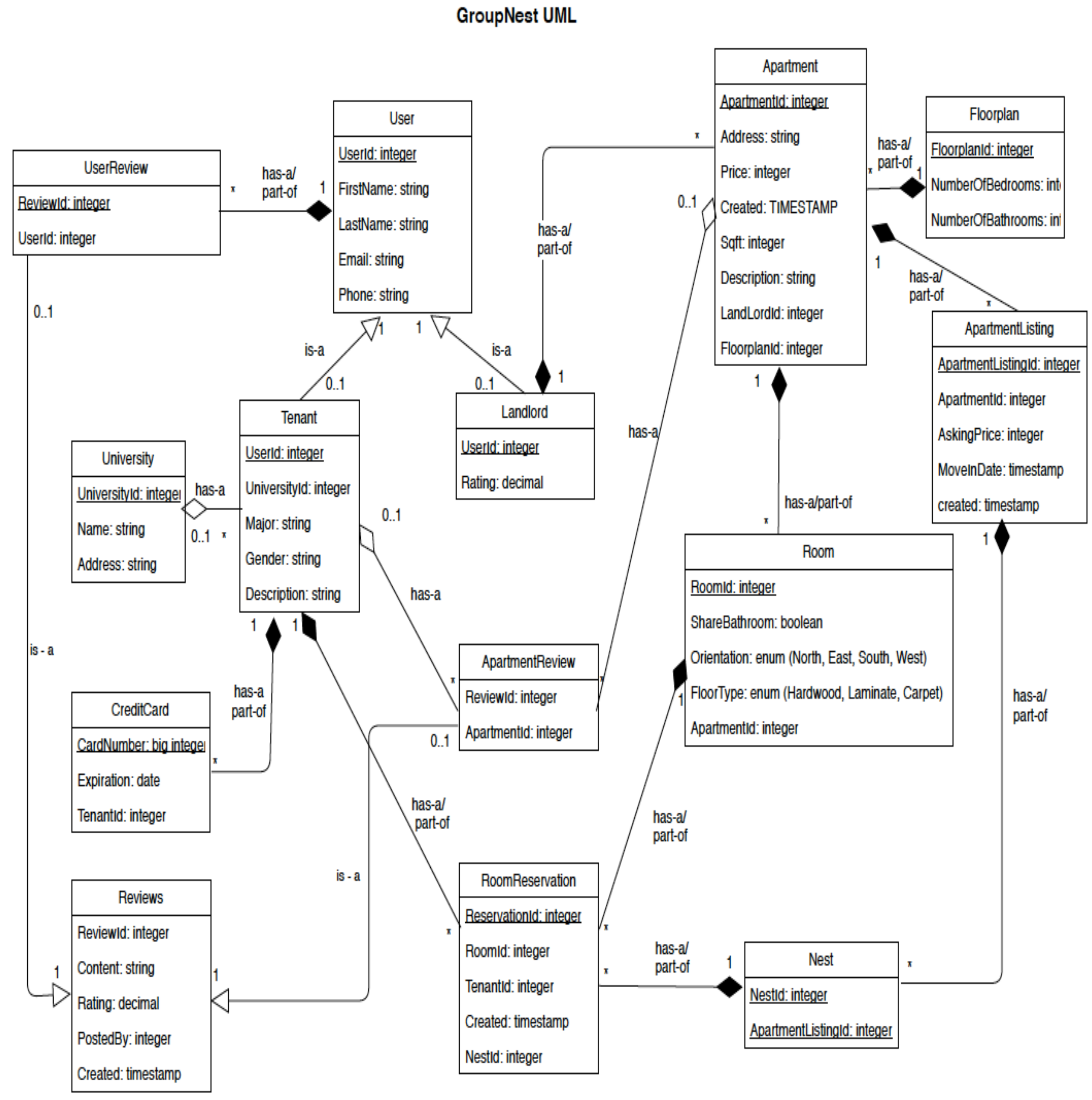
```
SELECT University.UniversityId,University.Name, TENANT_RESERVATION.Tenant_CNT

FROM University

        INNER JOIN

        (SELECT Tenant.UniversityId, COUNT(Tenant.UserId) AS Tenant_CNT
        FROM Tenant INNER JOIN RoomReservation
        ON Tenant.UserId  = RoomReservation.TenantId
        GROUP BY UniversityId) AS TENANT_RESERVATION

        ON University.UniversityId = TENANT_RESERVATION.UniversityId

GROUP BY University.UniversityId

ORDER BY TENANT_RESERVATION.Tenant_CNT DESC

LIMIT 10;
```

Result:



UniversityId	Name	Tenant_CNT
218	VA Boston Healthcare System - West Roxbury	70
254	New England School of Acupuncture Inc	66
299	Lakes Region Community College	63
611	Blanton-Peale Institute	60
287	Rivier University	59
311	University of New England	59
200	Wentworth Institute of Technology	59
353	Saint Michaels College	58
242	New England School of Photography	58
212	Jupiter Beauty Academy	58

Part2: Update UML



Part3: Update table:

For Milestone 3, we made some modification on tables as following:

1. normalized all tables that violated 1NF, 2NF and 3NF, to make sure there is no duplicates of information in multiple tables.
2. abstracted review with common information from userReview and ApartmentReview classes. maintain the generalization relationship between review and two subclasses.
3. modified constraint in tables to make the relationship much more reasonable.

```
# Create the schema if necessary.
CREATE SCHEMA IF NOT EXISTS GruopNest;
USE GruopNest;

# Drop tables if necessary.
DROP TABLE IF EXISTS CreditCard;
DROP TABLE IF EXISTS UserReview;
DROP TABLE IF EXISTS ApartmentReview;
DROP TABLE IF EXISTS Review;
DROP TABLE IF EXISTS RoomReservation;
DROP TABLE IF EXISTS Nest;
DROP TABLE IF EXISTS ApartmentListing;
DROP TABLE IF EXISTS Room;
DROP TABLE IF EXISTS Apartment;
DROP TABLE IF EXISTS FloorPlan;
DROP TABLE IF EXISTS Landlord;
DROP TABLE IF EXISTS Tenant;
DROP TABLE IF EXISTS University;
DROP TABLE IF EXISTS User;

# Create tables if necessary.
CREATE TABLE User (
UserId   INT UNSIGNED NOT NULL AUTO_INCREMENT,
FirstName VARCHAR(255) NOT NULL,
LastName VARCHAR(255) NOT NULL,
Email VARCHAR(255),
CONSTRAINT pk_User_UserId
  PRIMARY KEY (UserId)
);

CREATE TABLE CreditCard (
CardNumber VARCHAR(19) ,#NOT NULL, # max number of credit card digits is 19
ExpirationDate DATE,#NOT NULL, # time is not needed for exp date
UserId INT UNSIGNED,
CONSTRAINT pk_CreditCard_CardNumber
  PRIMARY KEY (CardNumber),
CONSTRAINT fk_CreditCard_User_UserId
  FOREIGN KEY (UserId)
  REFERENCES User (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE # row updated/deleted if data in parent table
updated/deleted
);


CREATE TABLE University (
UniversityId INT UNSIGNED NOT NULL AUTO_INCREMENT,
Name VARCHAR(255) NOT NULL,
Address VARCHAR(255),
City VARCHAR(255),
State VARCHAR(255),
Zip VARCHAR(255) NOT NULL,
CONSTRAINT pk_University_UniversityId
  PRIMARY KEY (UniversityId)
);

CREATE TABLE Tenant (
UserId INT UNSIGNED NOT NULL,
UniversityId INT UNSIGNED NOT NULL,
Major VARCHAR(255) NOT NULL,
Gender ENUM('Male', 'Female', 'Brand', 'Unknown') NOT NULL, # 3 options to choose from + NULL for
unknown
Bio TEXT, # anything you want to share about yourself
CONSTRAINT pk_Tenant_UserId
  PRIMARY KEY(UserId),
CONSTRAINT fk_Tenant_UserId
  FOREIGN KEY (UserId)
  REFERENCES User (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_Tenant_University_UniversityId
  FOREIGN KEY (UniversityId)
  REFERENCES University (UniversityId)
  ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Landlord (
UserId INT UNSIGNED NOT NULL,
CONSTRAINT pk_Landlord_UserId
  PRIMARY KEY(UserId),
CONSTRAINT fk_Landlord_UserId
  FOREIGN KEY (UserId)
  REFERENCES User (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE FloorPlan (
FloorPlanId INT UNSIGNED NOT NULL AUTO_INCREMENT,
NumberOfBedrooms INT NOT NULL,
NumberOfBathrooms INT NOT NULL,
CONSTRAINT pk_FloorPlan_FloorPlanId
  PRIMARY KEY(FloorPlanId)
);
```

```
CREATE TABLE Apartment (
ApartmentId INT UNSIGNED NOT NULL AUTO_INCREMENT,
FloorPlanId INT UNSIGNED NOT NULL,
Address VARCHAR(255),
City VARCHAR(255),
State VARCHAR(255),
Zip VARCHAR(255) NOT NULL,
Sqft INT UNSIGNED,
Name VARCHAR(255),
Description TEXT,
OwnerId INT UNSIGNED NOT NULL,
CONSTRAINT pk_Apartment_ApartmentId
  PRIMARY KEY(ApartmentId),
CONSTRAINT fk_Apartment_FloorPlan_FloorPlanId
  FOREIGN KEY (FloorPlanId)
  REFERENCES FloorPlan (FloorPlanId)
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_Apartment_Landlord_OwnerId
  FOREIGN KEY (OwnerId)
  REFERENCES Landlord (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE # if landlord is deleted, appts are deleted
);

CREATE TABLE Room (
RoomId INT UNSIGNED NOT NULL AUTO_INCREMENT,
ApartmentId INT UNSIGNED NOT NULL,
Sqrt INT UNSIGNED,
RoomType ENUM('Master Bedroom', 'Guest Bedroom', 'Other') NOT NULL,
ShareBathroom BOOLEAN,
FloorType ENUM('Hardwood', 'Laminate', 'Carpet', 'Other'),
Description TEXT,
CONSTRAINT pk_Room_RoomId
  PRIMARY KEY(RoomId),
CONSTRAINT fk_Room_Apartment_ApartmentId
  FOREIGN KEY (ApartmentId)
  REFERENCES Apartment (ApartmentId)
  ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE ApartmentListing (
ListingId INT UNSIGNED NOT NULL AUTO_INCREMENT,
ApartmentId INT UNSIGNED NOT NULL,
Title VARCHAR(255),  #NOT NULL,
AskingPrice DECIMAL(13,2) NOT NULL,
MoveInDate DATE NOT NULL,
LeaseTermInDays  INT,# UNSIGNED NOT NULL,
Content TEXT,# NOT NULL,
Contact VARCHAR(255),# NOT NULL, # required, either phone or email
IsClosed BOOLEAN NOT NULL, # if closed, not shown to public, not available for lease
PostedBy INT UNSIGNED, #NOT NULL,
PostedDateTime TIMESTAMP,# NOT NULL, # for first time listing, later modification on the listing not
changing this value
LastModifiedDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT pk_Listing_ListingId
  PRIMARY KEY(ListingId),
CONSTRAINT fk_Listing_Apartment_ApartmentId
  FOREIGN KEY (ApartmentId)
  REFERENCES Apartment (ApartmentId)
  ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_Listing_Landlord_PostedBy
  FOREIGN KEY (PostedBy)
  REFERENCES Landlord (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE # if user is deleted, listings are deleted
);


CREATE TABLE Nest (
NestId INT UNSIGNED NOT NULL AUTO_INCREMENT,
ListingId INT UNSIGNED NOT NULL,
CreatedBy INT UNSIGNED,
CreationDateTime TIMESTAMP,# NOT NULL, # for first time creation, later modification not changing this
value
IsDeleted TINYINT(1) NOT NULL, # if deleted, all room reservations are deleted. if no nest is created under a
listing, a new nest has to be created in order to put reservations.
IsAcceptedByLandlord TINYINT(1) NOT NULL, # multiple nests under one listing is possible, the full nest
has the highest possibility to be accepted by the landlord
LastModifiedDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT pk_Nest_NestId
  PRIMARY KEY(NestId),
CONSTRAINT fk_Nest_Listing_ListingId
  FOREIGN KEY (ListingId)
  REFERENCES ApartmentListing (ListingId)
  ON UPDATE CASCADE ON DELETE CASCADE, # if listing is deleted, nests are deleted
CONSTRAINT fk_Nest_Tenant_CreatedBy
  FOREIGN KEY (CreatedBy)
  REFERENCES Tenant (UserId)
  ON UPDATE CASCADE ON DELETE SET NULL
);

CREATE TABLE RoomReservation (
ReservationId INT UNSIGNED NOT NULL AUTO_INCREMENT,
RoomId INT UNSIGNED NOT NULL,
TenantId INT UNSIGNED NOT NULL,
ReservationDateTime TIMESTAMP, # for first time reservation, later modification on the reservation not
changing this value
NestId INT UNSIGNED NOT NULL,
OfferedPrice DECIMAL(13,2), # a negotiable price that the tenant is willing to offer. should be lower than the
apartment listing price
Contact VARCHAR(255), # contact for negotiation
IsCancelled TINYINT(1), # if cancelled, room under the nest can still be reserved by others, but at any time,
only one active reservation for one room is allowed
LastModifiedDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT pk_RoomReservation_ReservationId
  PRIMARY KEY(ReservationId),
CONSTRAINT uq_RoomReservation_Reserve
  UNIQUE KEY(RoomId, NestId),
CONSTRAINT fk_RoomReservation_Room_RoomId
  FOREIGN KEY (RoomId)
  REFERENCES Room (RoomId)
  ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_RoomReservation_Tenant_TenantId
  FOREIGN KEY (TenantId)
  REFERENCES Tenant (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_RoomReservation_Nest_NestId
  FOREIGN KEY (NestId)
  REFERENCES Nest (NestId)
  ON UPDATE CASCADE ON DELETE CASCADE # if nest is deleted, reservations are deleted
);


CREATE TABLE Review (
ReviewId INT UNSIGNED NOT NULL AUTO_INCREMENT,
PostedBy INT UNSIGNED,
PostedDateTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, # for first time review,
later modification not changing this value
LastModifiedDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
Content TEXT,
Rating DECIMAL(2,1) NOT NULL, # 2 digits precision and 1 decimal digits for 0.0 to 5.0 rating range
IsDeleted TINYINT(1), # if poster decided to have it deleted, not shown/calculated into user's average rating
CONSTRAINT pk_Review_ReviewId
  PRIMARY KEY(ReviewId),
CONSTRAINT fk_Review_User_PostedBy
  FOREIGN KEY (PostedBy)
  REFERENCES User (UserId)
  ON UPDATE CASCADE ON DELETE SET NULL
);


CREATE TABLE UserReview (
ReviewId INT UNSIGNED NOT NULL AUTO_INCREMENT,
UserId INT UNSIGNED NOT NULL, # either tenant or landlord. tenant can review tenant and landlord;
landlord can review tenant
Type ENUM('Tenant','Landlord') NOT NULL,
CONSTRAINT pk_UserReview_ReviewId
  PRIMARY KEY(ReviewId),
CONSTRAINT fk_UserReview_ReviewId
  FOREIGN KEY(ReviewId)
  REFERENCES Review(ReviewId)
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_UserReview_User_UserId
  FOREIGN KEY (UserId)
  REFERENCES User (UserId)
  ON UPDATE CASCADE ON DELETE CASCADE
);


CREATE TABLE ApartmentReview (
ReviewId INT UNSIGNED NOT NULL AUTO_INCREMENT,
ApartmentId INT UNSIGNED,
CONSTRAINT pk_ApartmentReview_ReviewId
  PRIMARY KEY(ReviewId),
CONSTRAINT fk_ApartmentReview_ReviewId
  FOREIGN KEY(ReviewId)
  REFERENCES Review(ReviewId)
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_ApartmentReview_Apartment_ApartmentId
  FOREIGN KEY (ApartmentId)
  REFERENCES Apartment (ApartmentId)
  ON UPDATE CASCADE ON DELETE CASCADE
  )
```