Database Systems

Inefficiency and Solution at Marist College

Make Marist Great Again

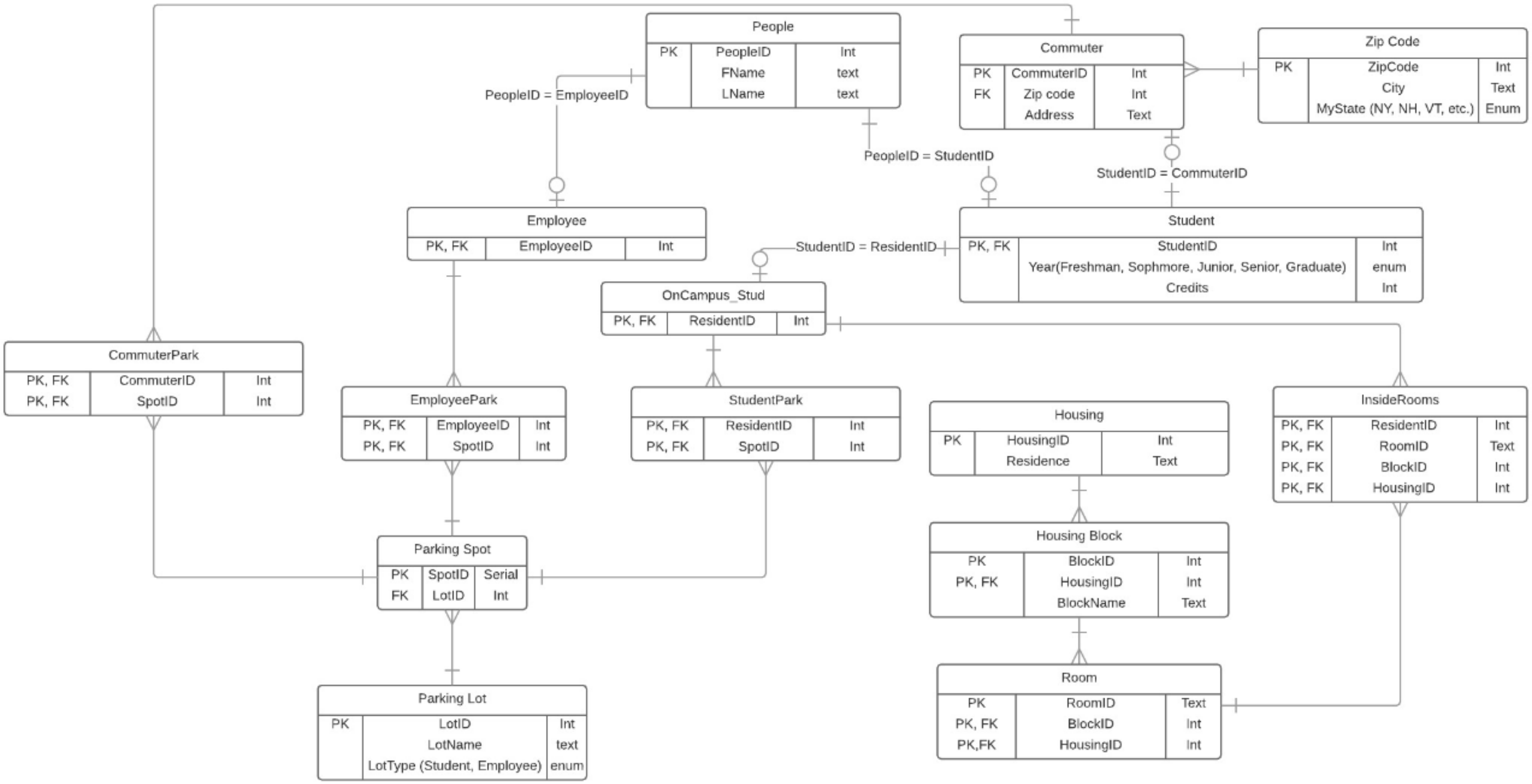
Table of Contents

Executive Summary

Marist College has approximately 5,000 undergraduate and graduate students that attend campus daily. On top of all of these students there are also employees; these employees include professors, staff, and consultants. Each person attending Marist College is given a parking pass that correlates to ones credits, or building that they work in. Marist College provides a very inefficient strategy to assign parking, where many faculty and students are placed in parking lots that are not near their designated location. The proposed system will provide Marist with a database that will better Marist's parking availability.

This database will contain sample data that was retrieved directly from Marist's student body. With this data, it will be seen that there are students parking in lots that does not logically make sense. Implementation will include self-assigning triggers to provide the most efficient parking strategy for Marist's student body and employees.

Entity Relationship Diagram



PeopleID = EmployeeID

PeopleID = StudentID

StudentID = CommuterID

StudentID = ResidentID

People

PK	PeopleID	Int
	FName	text
	LName	text

Commuter

PK	CommuterID	Int
FK	Zip code	Int
	Address	Text

Zip Code

PK	ZipCode	Int
	City	Text
	MyState (NY, NH, VT, etc.)	Enum

Employee

PK, FK	EmployeeID	Int

Student

PK, FK	StudentID	Int
	Year(Freshman, Sophmore, Junior, Senior, Graduate)	enum
	Credits	Int

OnCampus_Stud

PK, FK	ResidentID	Int

CommuterPark

PK, FK	CommuterID	Int
PK, FK	SpotID	Int

EmployeePark

PK, FK	EmployeeID	Int
PK, FK	SpotID	Int

StudentPark

PK, FK	ResidentID	Int
PK, FK	SpotID	Int

Housing

PK	HousingID	Int
	Residence	Text

InsideRooms

PK, FK	ResidentID	Int
PK, FK	RoomID	Text
PK, FK	BlockID	Int
PK, FK	HousingID	Int

Parking Spot

PK	SpotID	Serial
FK	LotID	Int

Housing Block

PK	BlockID	Int
PK, FK	HousingID	Int
	BlockName	Text

Parking Lot

PK	LotID	Int
	LotName	text
	LotType (Student, Employee)	enum

Room

PK	RoomID	Text
PK, FK	BlockID	Int
PK,FK	HousingID	Int

Tables

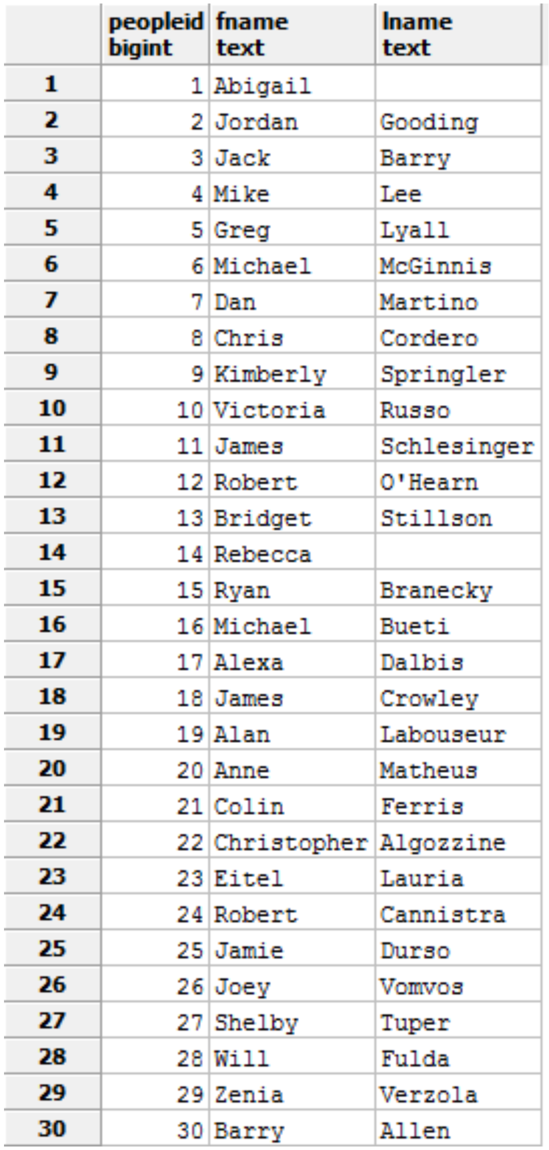
People: This table shows the first name, last name, and the unique identifier of each person.

```
CREATE TABLE IF NOT EXISTS People (
        PeopleID BIGINT NOT NULL UNIQUE,
        FName TEXT,
        LNAME TEXT,
        PRIMARY KEY(PeopleID)
);
```

Functional Dependencies

PeopleID → FName, LName

Sample data from the table **People:**



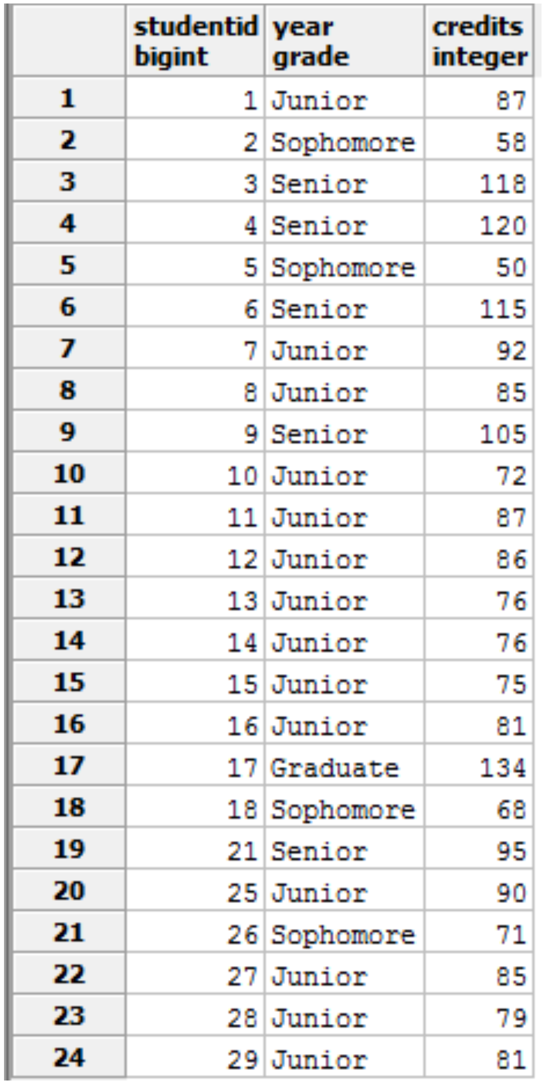
	peopleid bigint	fname text	lname text
1	1	Abigail	
2	2	Jordan	Gooding
3	3	Jack	Barry
4	4	Mike	Lee
5	5	Greg	Lyall
6	6	Michael	McGinnis
7	7	Dan	Martino
8	8	Chris	Cordero
9	9	Kimberly	Springler
10	10	Victoria	Russo
11	11	James	Schlesinger
12	12	Robert	O'Hearn
13	13	Bridget	Stillson
14	14	Rebecca	
15	15	Ryan	Branecky
16	16	Michael	Bueti
17	17	Alexa	Dalbis
18	18	James	Crowley
19	19	Alan	Labouseur
20	20	Anne	Matheus
21	21	Colin	Ferris
22	22	Christopher	Algozzine
23	23	Eitel	Lauria
24	24	Robert	Cannistra
25	25	Jamie	Durso
26	26	Joey	Vomvos
27	27	Shelby	Tuper
28	28	Will	Fulda
29	29	Zenia	Verzola
30	30	Barry	Allen

Student: This table contains people who are students, contains studentID, their year, and their credits.

```
CREATE TABLE IF NOT EXISTS Student (
        StudentID BIGINT NOT NULL UNIQUE REFERENCES People(PeopleID),
        Year grade,
        Credits INT,
        PRIMARY KEY(StudentID)
);
```

Function Dependencies:
StudentID → Year, Credits

Sample data from the table **Students**:



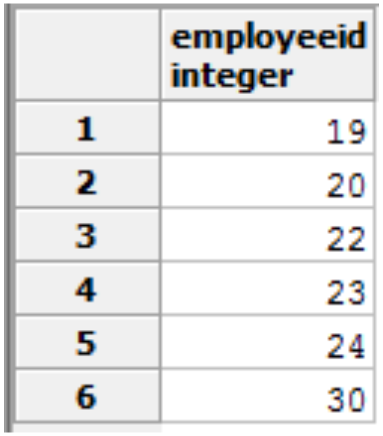
	studentid	year	credits
	bigint	grade	integer
1	1	Junior	87
2	2	Sophomore	58
3	3	Senior	118
4	4	Senior	120
5	5	Sophomore	50
6	6	Senior	115
7	7	Junior	92
8	8	Junior	85
9	9	Senior	105
10	10	Junior	72
11	11	Junior	87
12	12	Junior	86
13	13	Junior	76
14	14	Junior	76
15	15	Junior	75
16	16	Junior	81
17	17	Graduate	134
18	18	Sophomore	68
19	21	Senior	95
20	25	Junior	90
21	26	Sophomore	71
22	27	Junior	85
23	28	Junior	79
24	29	Junior	81

Employee: This table contains people who are students, contains studentID, their year, and their credits.

```
CREATE TABLE IF NOT EXISTS Employee (
	EmployeeID INT NOT NULL REFERENCES People(PeopleID),
	PRIMARY KEY(EmployeeID)
);
```

Functional Dependencies:
EmployeeID →

Sample data provides employeeID from the table **Employee.**



	employeeid integer
1	19
2	20
3	22
4	23
5	24
6	30

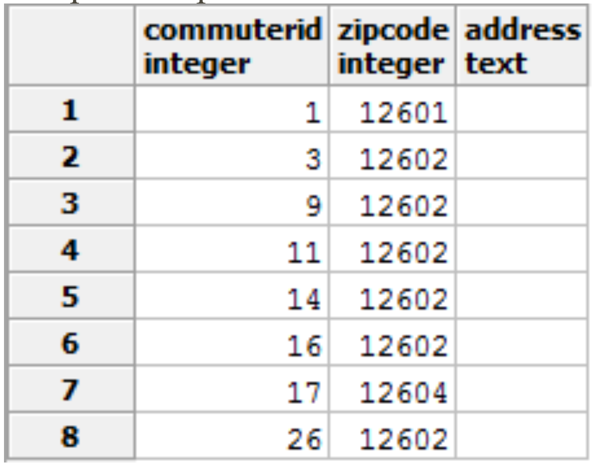
Commuter: This table contains students (people) who are commuters, contains CommuterID, their zipcode, and their address.

```
CREATE TABLE IF NOT EXISTS Commuter (
	CommuterID INT NOT NULL REFERENCES People(PeopleID),
	ZipCode INT NOT NULL REFERENCES ZipCode(ZipCode),
	Address TEXT,
	PRIMARY KEY(CommuterID)
);
```

Functional Dependencies:

CommuterID → ZipCode, Address

Sample data provides attributes from the table **Commuter:**



	commuterid integer	zipcode integer	address text
1	1	12601	
2	3	12602	
3	9	12602	
4	11	12602	
5	14	12602	
6	16	12602	
7	17	12604	
8	26	12602	

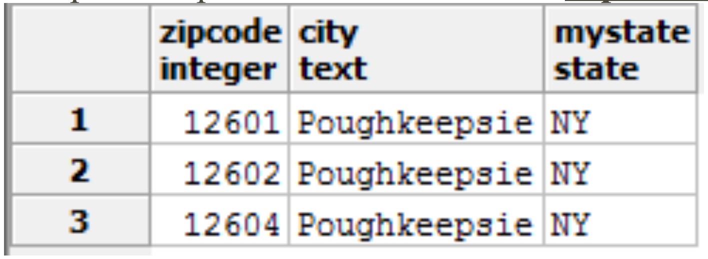
ZipCode: This table contains commuters address information, such as zip code, city, and state.

```
CREATE TABLE IF NOT EXISTS ZipCode (
	ZipCode INT NOT NULL,
	City TEXT,
	MyState State,
	PRIMARY KEY(ZipCode)
);
```

Functional Dependencies:

ZipCode → City, Mystate

Sample data provides attributes from **ZipCode:**



	zipcode integer	city text	mystate state
1	12601	Poughkeepsie	NY
2	12602	Poughkeepsie	NY
3	12604	Poughkeepsie	NY

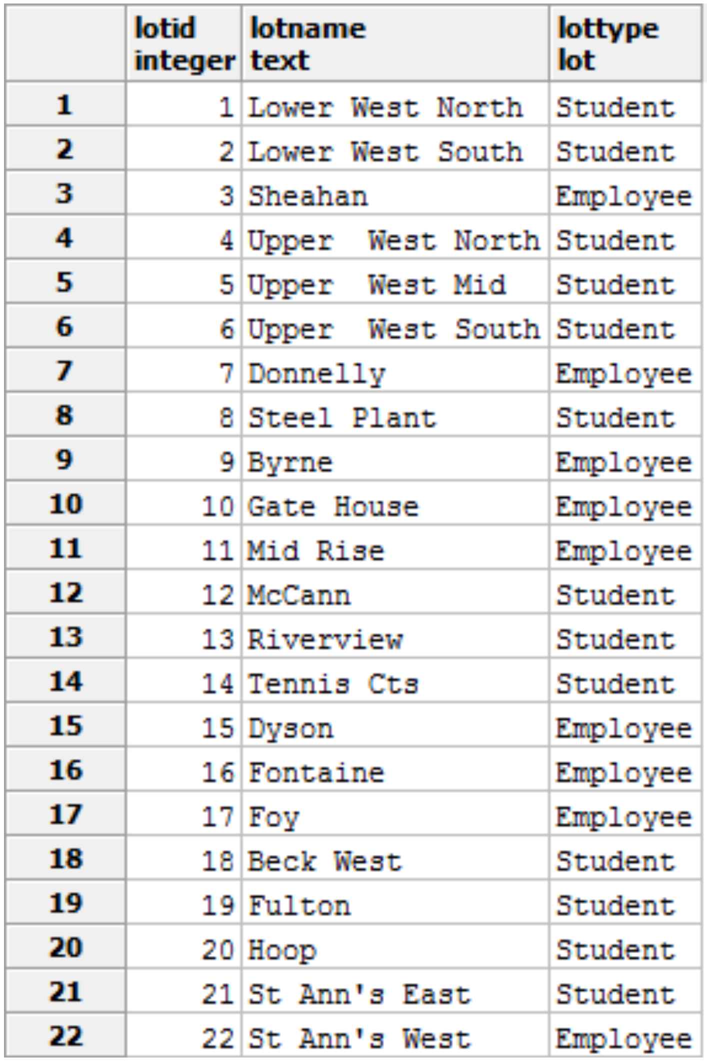
ParkingLot: This table contains each parking lot at Marist College; the attributes of this table includes LotID, LotName, and LotType.

```
CREATE TABLE IF NOT EXISTS ParkingLot (
	LotID INT NOT NULL UNIQUE,
	LotName TEXT,
	LotType lot,
	PRIMARY KEY(LotID)
);
```

Functional Dependencies:

LotID → LotName, LotType

Sample data provides attributes from **ParkingLot:**



	lotid integer	lotname text	lottype lot
1	1	Lower West North	Student
2	2	Lower West South	Student
3	3	Sheahan	Employee
4	4	Upper West North	Student
5	5	Upper West Mid	Student
6	6	Upper West South	Student
7	7	Donnelly	Employee
8	8	Steel Plant	Student
9	9	Byrne	Employee
10	10	Gate House	Employee
11	11	Mid Rise	Employee
12	12	McCann	Student
13	13	Riverview	Student
14	14	Tennis Cts	Student
15	15	Dyson	Employee
16	16	Fontaine	Employee
17	17	Foy	Employee
18	18	Beck West	Student
19	19	Fulton	Student
20	20	Hoop	Student
21	21	St Ann's East	Student
22	22	St Ann's West	Employee

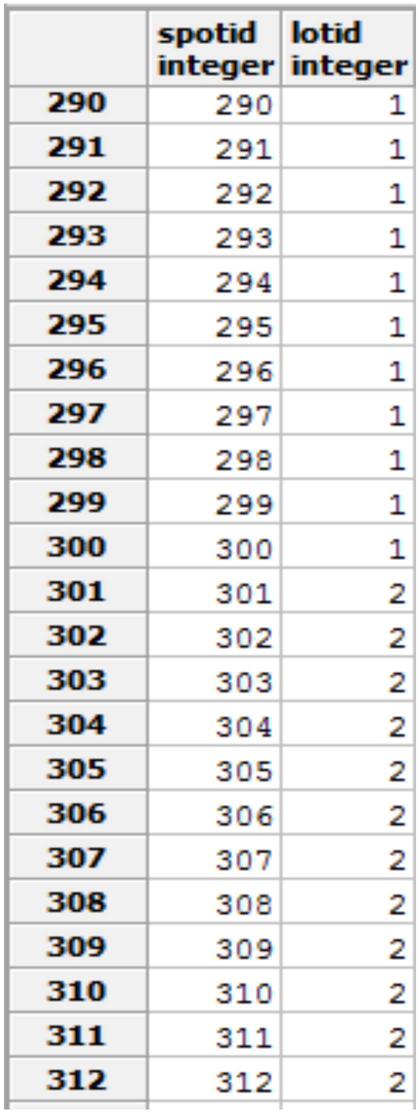
ParkingSpot: This table contains each parking spot at Marist College; the attributes of this table includes SpotID and LotID.

```
CREATE TABLE IF NOT EXISTS ParkingSpot (
        SpotID SERIAL NOT NULL UNIQUE,
        LotID INT NOT NULL REFERENCES ParkingLot(LotID),
        PRIMARY KEY(SpotID)
);
```

Functional Dependencies:

SpotID → LotID

Sample data provides attributes from the table **ParkingSpot:**



	spotid integer	lotid integer
290	290	1
291	291	1
292	292	1
293	293	1
294	294	1
295	295	1
296	296	1
297	297	1
298	298	1
299	299	1
300	300	1
301	301	2
302	302	2
303	303	2
304	304	2
305	305	2
306	306	2
307	307	2
308	308	2
309	309	2
310	310	2
311	311	2
312	312	2

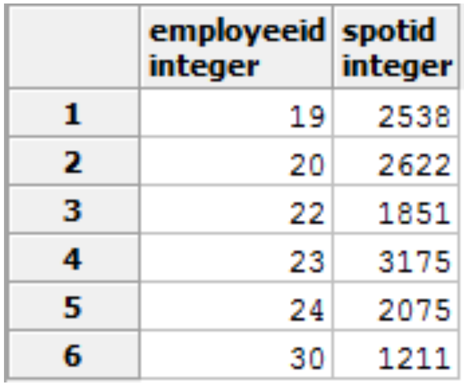
EmployeePark: This table contains each Employee's ID and their designated parking spot; the attributes of this table includes EmployeeID and SpotID.

```
CREATE TABLE IF NOT EXISTS EmployeePark (
	EmployeeID INT NOT NULL UNIQUE REFERENCES Employee(EmployeeID),
	SpotID INT NOT NULL UNIQUE REFERENCES ParkingSpot(SpotID),
	PRIMARY KEY(SpotID)
);
```

Functional Dependencies:

(EmployeeID, SpotID) →

Sample data provides attributes from the table **EmployeePark:**



	employeeid integer	spotid integer
1	19	2538
2	20	2622
3	22	1851
4	23	3175
5	24	2075
6	30	1211

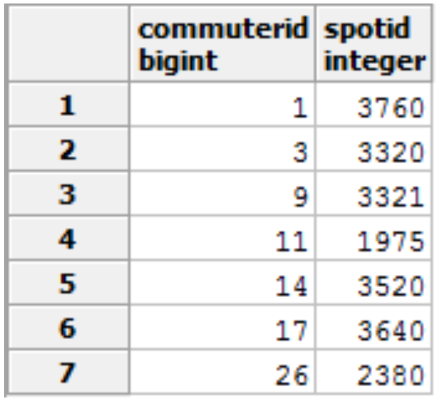
CommmuterPark: This table contains each commuter's ID and their designated parking spot; the attributes of this table includes CommuterID and SpotID.

```
CREATE TABLE IF NOT EXISTS CommuterPark (
        CommuterID BIGINT NOT NULL UNIQUE REFERENCES Commuter(CommuterID),
        SpotID INT NOT NULL UNIQUE REFERENCES ParkingSpot(SpotID),
        PRIMARY KEY(CommuterID, SpotID)
);
```

Functional Dependencies:

(CommuterID, SpotID) →

Sample data provides attributes from the table **CommuterPark:**



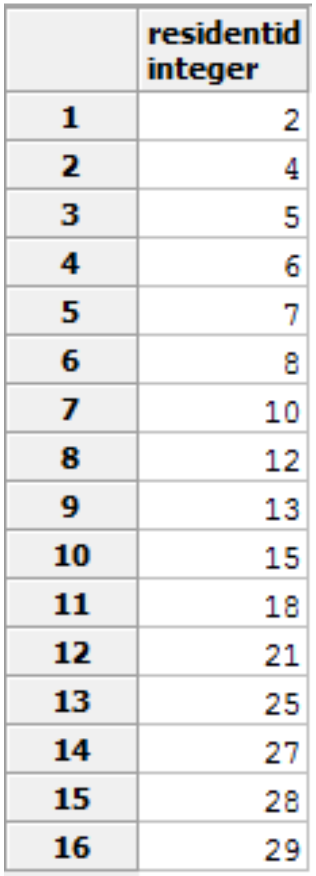
	commuterid bigint	spotid integer
1	1	3760
2	3	3320
3	9	3321
4	11	1975
5	14	3520
6	17	3640
7	26	2380

OnCampus_Stud: This table contains each students ID (ResidentID); the attributes of this table includes ResidentID.

```
CREATE TABLE IF NOT EXISTS OnCampus_Stud (
        ResidentID INT NOT NULL REFERENCES Student(StudentID),
        PRIMARY KEY(ResidentID)
);
```

Functional Dependencies:

ResidentID → Sample data provides attributes from the table **OnCampus_Stud:**



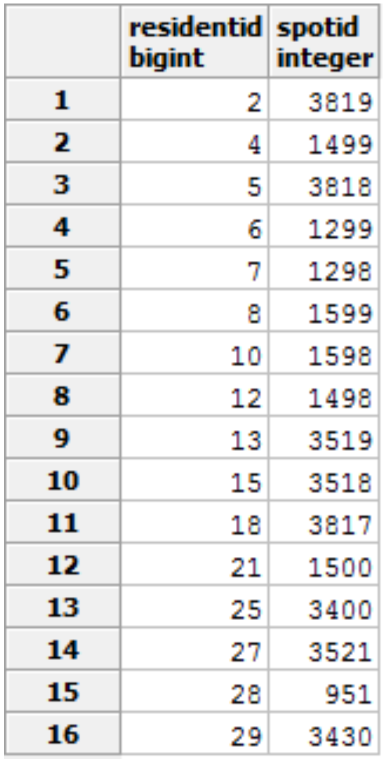
	residentid integer
1	2
2	4
3	5
4	6
5	7
6	8
7	10
8	12
9	13
10	15
11	18
12	21
13	25
14	27
15	28
16	29

StudentPark: This table contains each resident's ID and their designated parking spot; the attributes of this table includes ResidentID and SpotID.

```
CREATE TABLE IF NOT EXISTS StudentPark (
        ResidentID BIGINT NOT NULL UNIQUE REFERENCES OnCampus_Stud(ResidentID),
        SpotID INT NOT NULL UNIQUE REFERENCES ParkingSpot(SpotID),
        PRIMARY KEY(ResidentID, SpotID)
);
```

Functional Dependencies:
(ResidentID, SpotID) →

Sample data provides attributes from the table **StudentPark:**



	residentid bigint	spotid integer
1	2	3819
2	4	1499
3	5	3818
4	6	1299
5	7	1298
6	8	1599
7	10	1598
8	12	1498
9	13	3519
10	15	3518
11	18	3817
12	21	1500
13	25	3400
14	27	3521
15	28	951
16	29	3430

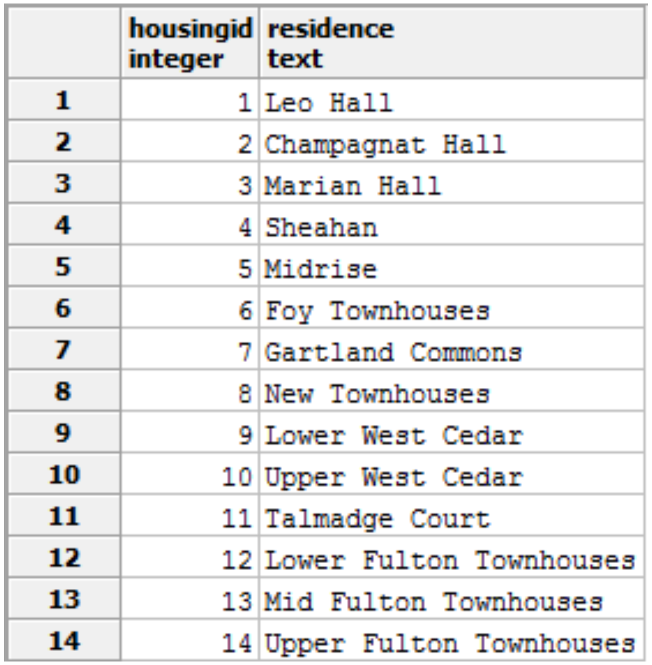
Housing: This table contains each residence and its ID; the attributes of this table includes HousingID and Residence.

```
CREATE TABLE IF NOT EXISTS Housing (
	HousingID SERIAL NOT NULL UNIQUE,
	Residence TEXT,
	PRIMARY KEY(HousingID)
);
```

Functional Dependencies:

HousingID → Residence

Sample data provides attributes from the table **Housing:**



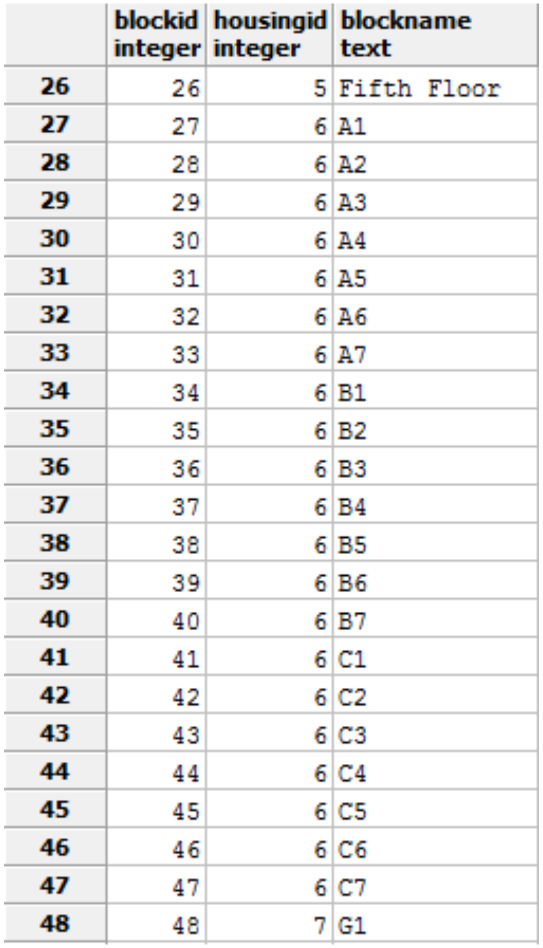
	housingid	residence
	integer	text
1	1	Leo Hall
2	2	Champagnat Hall
3	3	Marian Hall
4	4	Sheahan
5	5	Midrise
6	6	Foy Townhouses
7	7	Gartland Commons
8	8	New Townhouses
9	9	Lower West Cedar
10	10	Upper West Cedar
11	11	Talmadge Court
12	12	Lower Fulton Townhouses
13	13	Mid Fulton Townhouses
14	14	Upper Fulton Townhouses

HousingBlock: This table contains each housing area ID, as well as the blocks and IDs that are inside the housing areas; the attributes of this table includes BlockID, HousingID, and BlockName.

```
CREATE TABLE IF NOT EXISTS HousingBlock (
        BlockID INT NOT NULL,
        HousingID INT NOT NULL REFERENCES   Housing(HousingID),
        BlockName TEXT,
        PRIMARY KEY(BlockID, HousingID)
);
```

Functional Dependencies:
(BlockID, HousingID) → BlockName

Sample data provides attributes from the table **HousingBlock:**



	blockid integer	housingid integer	blockname text
26	26	5	Fifth Floor
27	27	6	A1
28	28	6	A2
29	29	6	A3
30	30	6	A4
31	31	6	A5
32	32	6	A6
33	33	6	A7
34	34	6	B1
35	35	6	B2
36	36	6	B3
37	37	6	B4
38	38	6	B5
39	39	6	B6
40	40	6	B7
41	41	6	C1
42	42	6	C2
43	43	6	C3
44	44	6	C4
45	45	6	C5
46	46	6	C6
47	47	6	C7
48	48	7	G1

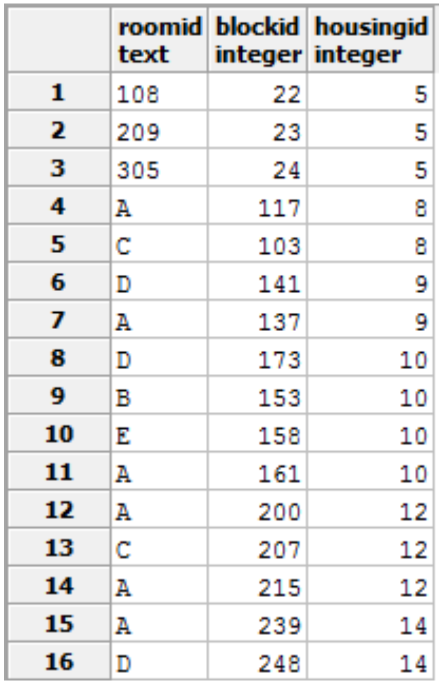
Room: This table contains each room and its ID that is occupied as well as the block ID and the housing area ID; the attributes of this table includes RoomID, BlockID, and HousingID.

```
CREATE TABLE IF NOT EXISTS Room (
            RoomID TEXT NOT NULL,
            BlockID INT NOT NULL,
            HousingID INT NOT NULL,
            PRIMARY KEY(RoomID, BlockID, HousingID),
            FOREIGN KEY(BlockID, HousingID) REFERENCES HousingBlock(BlockID, HousingID)

);
```

Functional Dependencies:

(RoomID, BlockID, HousingID) →

Sample data provides attributes from the table **Room:**



	roomid text	blockid integer	housingid integer
1	108	22	5
2	209	23	5
3	305	24	5
4	A	117	8
5	C	103	8
6	D	141	9
7	A	137	9
8	D	173	10
9	B	153	10
10	E	158	10
11	A	161	10
12	A	200	12
13	C	207	12
14	A	215	12
15	A	239	14
16	D	248	14

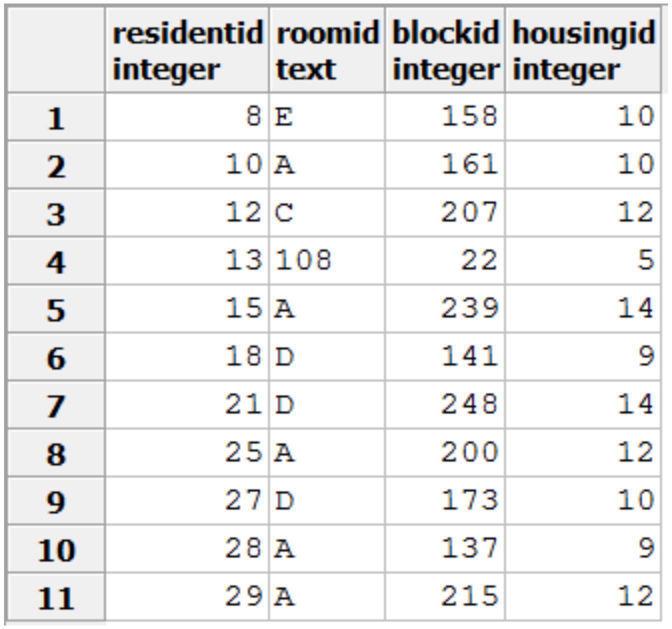
InsideRoom: This table contains each resident's ID, their room ID, block ID and housing area ID; the attributes of this table includes ResidentID, RoomID, BlockID, and HousingID.

```
CREATE TABLE IF NOT EXISTS InsideRooms (
        ResidentID INT NOT NULL UNIQUE REFERENCES OnCampus_Stud(ResidentID),
        RoomID TEXT NOT NULL,
        BlockID INT NOT NULL,
        HousingID INT NOT NULL,
        PRIMARY KEY(ResidentID, RoomID, BlockID, HousingID),
        FOREIGN KEY(RoomID, BlockID, HousingID) REFERENCES Room(RoomID, BlockID, HousingID)
);
```

Functional Dependencies:

(ResidentID, RoomID, BlockID, HousingID) →

Sample data provides attributes from the table **InsideRoom:**

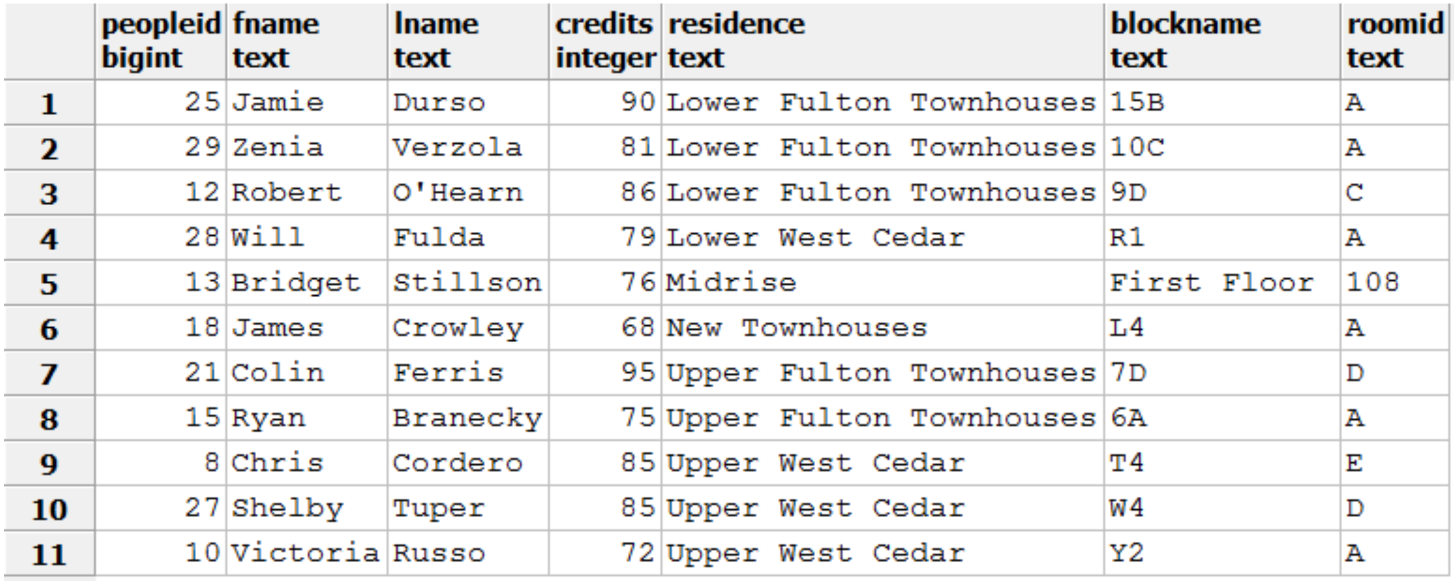


	residentid	roomid	blockid	housingid
	integer	text	integer	integer
1	8	E	158	10
2	10	A	161	10
3	12	C	207	12
4	13	108	22	5
5	15	A	239	14
6	18	D	141	9
7	21	D	248	14
8	25	A	200	12
9	27	D	173	10
10	28	A	137	9
11	29	A	215	12

OnCampus_Stud_Info:

This view was created to show each student that is on campus, along with their earned credits and housing assignments.

```
create view OnCampus_Stud_Info
as
select p.peopleid, p.fname, p.lname, s.credits, h.residence, b.blockname, i.roomID
from people p inner join student s on p.peopleid = s.studentid
          inner join insiderooms i on p.peopleid = i.residentid
          inner join housing h on i.housingid = h.housingid
          inner join housingblock b on i.housingid = b.housingid
          where b.blockid = i.blockid
          order by residence  ;
```

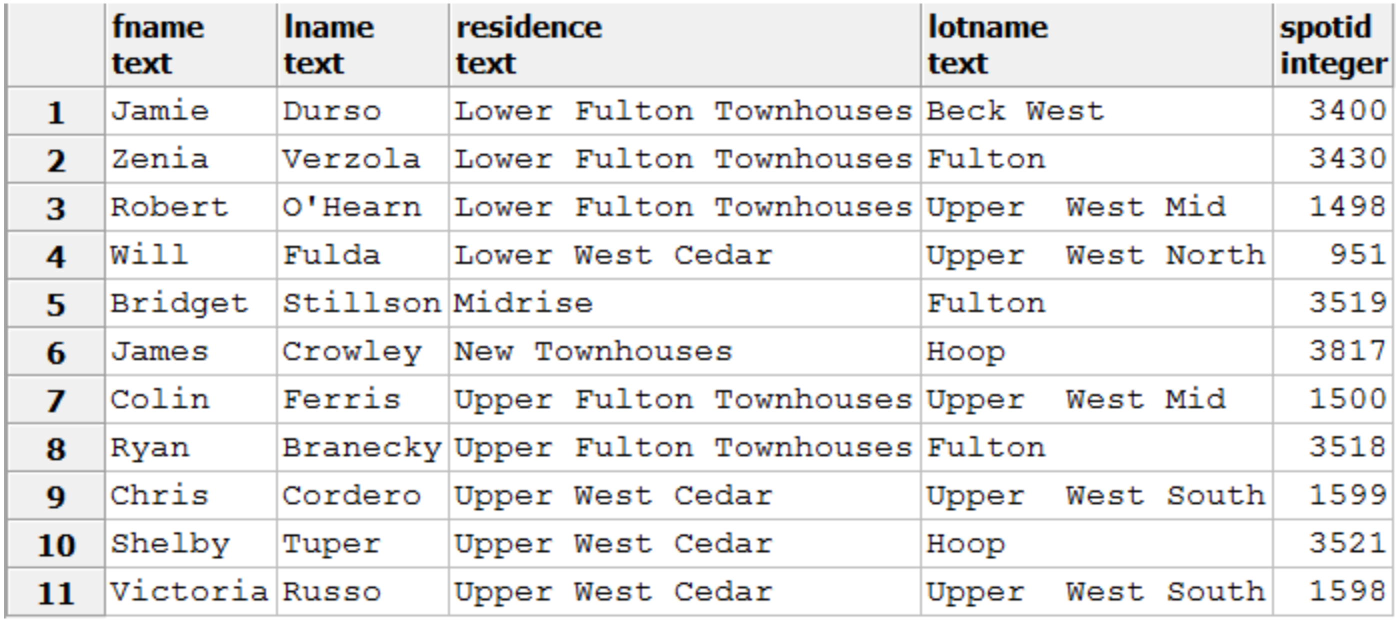


	peopleid	fname	lname	credits	residence	blockname	roomid
	bigint	text	text	integer	text	text	text
1	25	Jamie	Durso	90	Lower Fulton Townhouses	15B	A
2	29	Zenia	Verzola	81	Lower Fulton Townhouses	10C	A
3	12	Robert	O'Hearn	86	Lower Fulton Townhouses	9D	C
4	28	Will	Fulda	79	Lower West Cedar	R1	A
5	13	Bridget	Stillson	76	Midrise	First Floor	108
6	18	James	Crowley	68	New Townhouses	L4	A
7	21	Colin	Ferris	95	Upper Fulton Townhouses	7D	D
8	15	Ryan	Branecky	75	Upper Fulton Townhouses	6A	A
9	8	Chris	Cordero	85	Upper West Cedar	T4	E
10	27	Shelby	Tuper	85	Upper West Cedar	W4	D
11	10	Victoria	Russo	72	Upper West Cedar	Y2	A

StudentParking_Info:

This view was created to show where each student is living on campus in comparison to where they park on campus. This allows the inefficient parking at Marist College to be exposed.

```
create view StudentParking_Info
as
select o.fname, o.lname, o.residence, lot.lotname, studpark.spotid
from oncampus_stud_info o inner join studentpark studpark on o.peopleid = studpark.residentid
                         inner join parkingspot spot on studpark.spotid = spot.spotid
                         inner join parkinglot lot on spot.lotid = lot.lotid
                         where spot.spotid = studpark.spotid ;
```

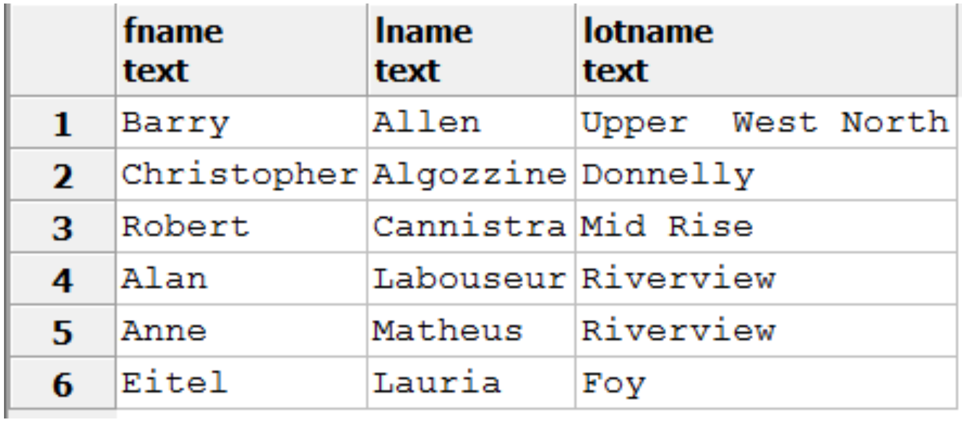


	fname text	lname text	residence text	lotname text	spotid integer
1	Jamie	Durso	Lower Fulton Townhouses	Beck West	3400
2	Zenia	Verzola	Lower Fulton Townhouses	Fulton	3430
3	Robert	O'Hearn	Lower Fulton Townhouses	Upper West Mid	1498
4	Will	Fulda	Lower West Cedar	Upper West North	951
5	Bridget	Stillson	Midrise	Fulton	3519
6	James	Crowley	New Townhouses	Hoop	3817
7	Colin	Ferris	Upper Fulton Townhouses	Upper West Mid	1500
8	Ryan	Branecky	Upper Fulton Townhouses	Fulton	3518
9	Chris	Cordero	Upper West Cedar	Upper West South	1599
10	Shelby	Tuper	Upper West Cedar	Hoop	3521
11	Victoria	Russo	Upper West Cedar	Upper West South	1598

Employee_Parking:

This view was created to show where each professor that in currently in the database is parking. Each teacher as of now is based out of Hancock; this table shows how the professors are spread throughout Marist's parking lots.

```
create view employee_parking
as
SELECT people.fname, people.lname, lotname
from people inner join employee on people.peopleid = employee.employeeid
        inner join employeepark on employee.employeeid = employeepark.employeeid
        inner join parkingspot spot on employeepark.spotid = spot.spotid
        inner join parkinglot lot on spot.lotid = lot.lotid ;
```

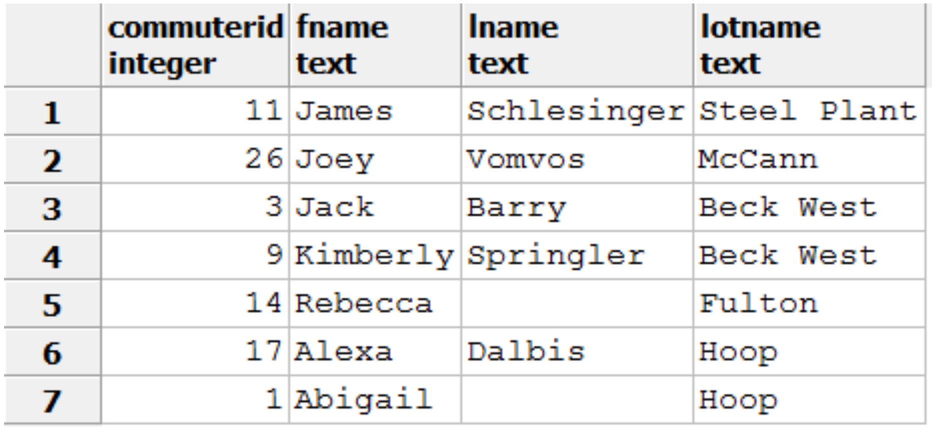


	fname text	lname text	lotname text
1	Barry	Allen	Upper West North
2	Christopher	Algozzine	Donnelly
3	Robert	Cannistra	Mid Rise
4	Alan	Labouseur	Riverview
5	Anne	Matheus	Riverview
6	Eitel	Lauria	Foy

Commuter_Parking:

This view was created to show where each commuter parks at Marist College. This table will be useful because in a future database, the user will be able to enter their classes, showing how close or far they have to park from their classroom.

```
create view commuter_parking
as
SELECT commuter.commuterid, people.fname, people.lname, lotname
from people inner join commuter on people.peopleid = commuter.commuterid
        inner join commuterpark on commuter.commuterid = commuterpark.commuterid
        inner join parkingspot spot on commuterpark.spotid = spot.spotid
        inner join parkinglot lot on spot.lotid = lot.lotid ;
```

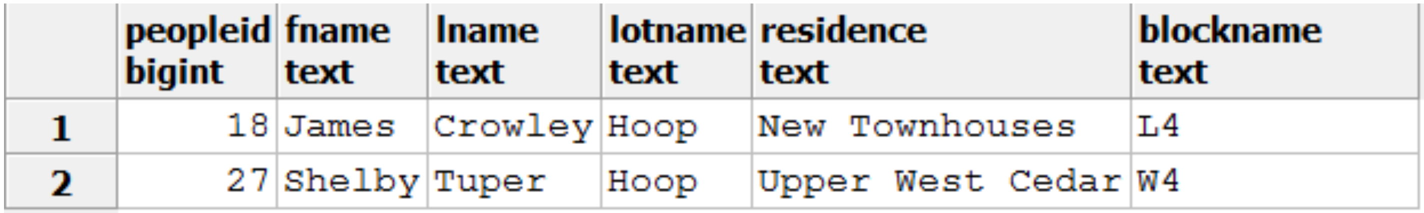


	commuterid	fname	lname	lotname
	integer	text	text	text
1	11	James	Schlesinger	Steel Plant
2	26	Joey	Vomvos	McCann
3	3	Jack	Barry	Beck West
4	9	Kimberly	Springler	Beck West
5	14	Rebecca		Fulton
6	17	Alexa	Dalbis	Hoop
7	1	Abigail		Hoop

Student_hoopParking:

This view was created as an example. This virtual table will show all of the students who specifically park in hoop and where they live. This could help determine whether people should live a specific lot based on where they live and their role at Marist (student).

```
create view student_hookParking
as
select studentparking_info.peopleid, studentparking_info.fname, studentparking_info.lname, studentparking_info.lotname,
OnCampus_Stud_Info.residence, OnCampus_Stud_Info.blockname
from studentparking_info
inner join OnCampus_Stud_Info on studentparking_info.peopleid = OnCampus_Stud_Info.peopleid
where studentparking_info.lotname = 'Hoop' ;
```

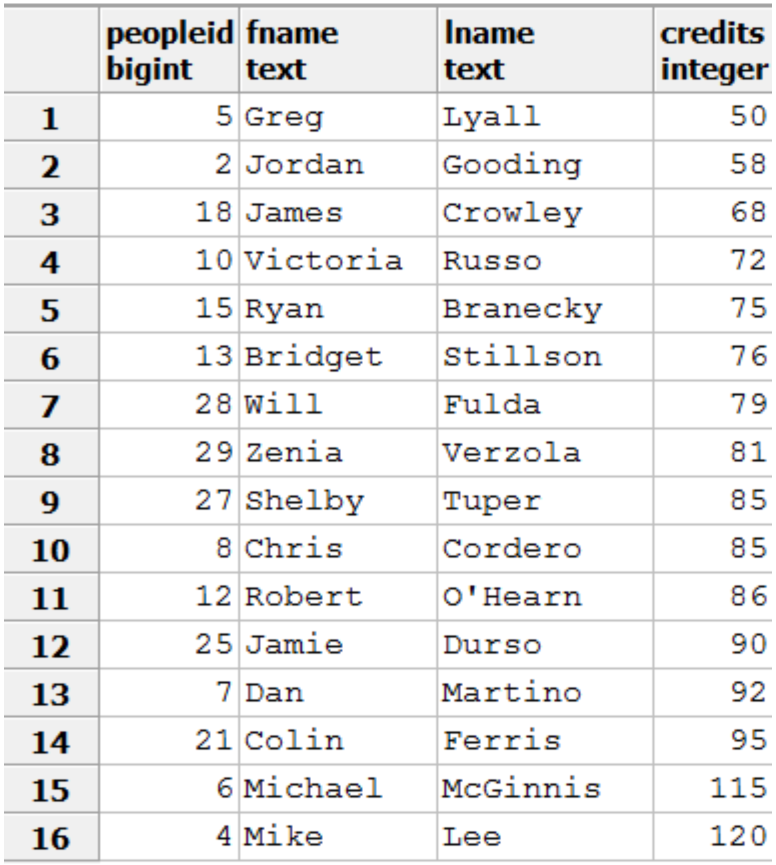


	peopleid	fname	lname	lotname	residence	blockname
	bigint	text	text	text	text	text
1	18	James	Crowley	Hoop	New Townhouses	L4
2	27	Shelby	Tuper	Hoop	Upper West Cedar	W4

Reports

Query returns the on campus students who have over 50 credits and are eligible to park on campus.

```
Select nc.peopleid, nc.fname, nc.lname, s.credits
from namecheck nc inner join people on nc.peopleid = people.peopleid
                inner join student s on people.peopleid = s.studentid
                inner join oncampus_stud o on s.studentid = o.residentid
where s.credits >= 50
order by credits
```

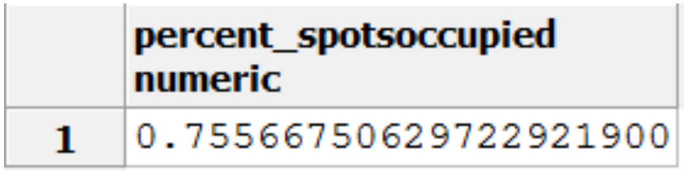


	peopleid bigint	fname text	lname text	credits integer
1	5	Greg	Lyall	50
2	2	Jordan	Gooding	58
3	18	James	Crowley	68
4	10	Victoria	Russo	72
5	15	Ryan	Branecky	75
6	13	Bridget	Stillson	76
7	28	Will	Fulda	79
8	29	Zenia	Verzola	81
9	27	Shelby	Tuper	85
10	8	Chris	Cordero	85
11	12	Robert	O'Hearn	86
12	25	Jamie	Durso	90
13	7	Dan	Martino	92
14	21	Colin	Ferris	95
15	6	Michael	McGinnis	115
16	4	Mike	Lee	120

Query returns the percentage of spots that are currently filled on Marist’s campus.

(Percentage is very small due to sample size. This can alert Marist when to potentially start to begin adding more parking options.)

```
SELECT (
       CAST(
          (SELECT COUNT(parkingspot.spotid) AS spotsTaken
                 FROM parkingspot full outer join commuterpark c on parkingspot.spotid = c.spotid
                               full outer join employeepark e on parkingspot.spotid = e.spotid
                               full outer join studentpark on parkingspot.spotid = studentpark.spotid
                               full outer join parkinglot on parkingspot.lotid = parkinglot.lotid
                 WHERE c.spotid = parkingspot.spotid or e.spotid = parkingspot.spotid or studentpark.spotid =
parkingspot.spotid

            ) as decimal(5,2)
           )
            /
            ( select count(spotid) as allSpots
              from parkingspot
            )
    * 100
         ) as    Percent_spotsOccupied ;
```

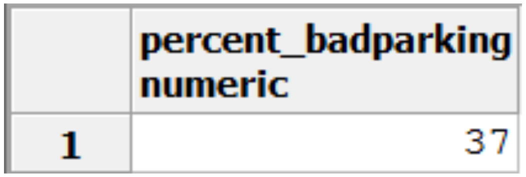


	percent_spotsoccupied numeric
1	0.75566750629722921900

Query returns the percentage of students that live in Lower Fulton Townhouses but do not have the ability to park in the Fulton Lot

(This shows an example of the inefficiency in regards to where students and their parking assignments.)

```
SELECT     TRUNC (
       CAST(
          (SELECT COUNT(peopleID) AS badParking
                FROM people inner join student on people.peopleid = student.studentid
                         inner join oncampus_stud o on people.peopleid = o.residentid
                         inner join studentpark s on o.residentid = s.residentid
                         inner join parkingspot spot on s.spotid = spot.spotid
                         inner join parkinglot lot on spot.lotid = lot.lotid
                         inner join insiderooms i on o.residentid = i.residentid
                         inner join room r on i.roomid = r.roomid
                         inner join housingblock b on r.blockid = b.blockid
                         inner join housing h on r.housingid = h.housingid
                WHERE lotname not like '%Fulton%' and residence = 'Lower Fulton Townhouses'

            ) as decimal(5,2)
           )
             /
            ( select count(studentid) as allStudents
              from student
             )
    * 100
         ) as    Percent_badParking ;
```

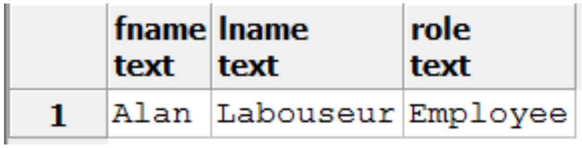


	percent_badparking numeric
1	37

Stored Procedures

This **stored procedure** shows what each person's in the database role is according to his or her PeopleID.

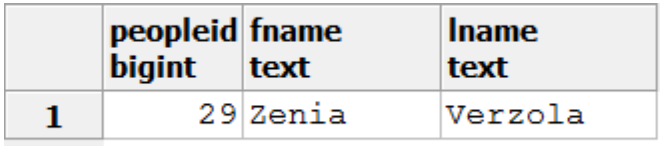
```
CREATE OR REPLACE FUNCTION role_at_marist(_identifier int)
  RETURNS TABLE (fname text, lname text, "role" text) AS
$func$
   SELECT p.fname, p.lname, text 'Commuter'
   FROM   people p
   WHERE  p.peopleid = $1
   AND    EXISTS (SELECT 1 FROM commuter c WHERE c.commuterid = p.peopleid)

   UNION ALL
   SELECT p.fname, p.lname, 'Employee'
   FROM   people p
   WHERE  p.peopleid = $1
   AND    EXISTS (SELECT 1 FROM employee e WHERE e.employeeid = p.peopleid)

   UNION ALL
   SELECT p.fname, p.lname, 'Resident'
   FROM   people p
   WHERE  p.peopleid = $1
   AND    EXISTS (SELECT 1 FROM studentpark s WHERE s.residentid = p.peopleid)
$func$ LANGUAGE sql;

SELECT * FROM role_at_marist(19);
```



	fname text	lname text	role text
1	Alan	Labouseur	Employee

This **stored procedure** takes an integer (spotid) as an input and shows the specific person’s first name, last name and their ID (peopleid) to which the spot belongs too.

```
create or replace function spotInfo(int, refcursor) returns refcursor as
$$
 declare
  info int              :=$1;
  resultset refcursor :=$2;
 begin
  open resultset for
   select distinct people.peopleid ,people.fname, people.lname
   from people full outer join namecheck on people.peopleid = namecheck.peopleid
               full outer join parkingspot on namecheck.spotid = parkingspot.spotid
               full outer join commuterpark on parkingspot.spotid = commuterpark.spotid
               full outer join employeepark on parkingspot.spotid = employeepark.spotid
               full outer join studentpark on parkingspot.spotid = studentpark.spotid
   where info = namecheck.spotid ;
  return resultset;
 end
$$
language plpgsql;

select spotInfo(3430, 'results');
fetch all from results;
```



	peopleid	fname	lname
	bigint	text	text
1	29	Zenia	Verzola

This **stored procedure** checks the most recent input (EmployeeID) into the Employee tables and assigns that input the next available parking space.

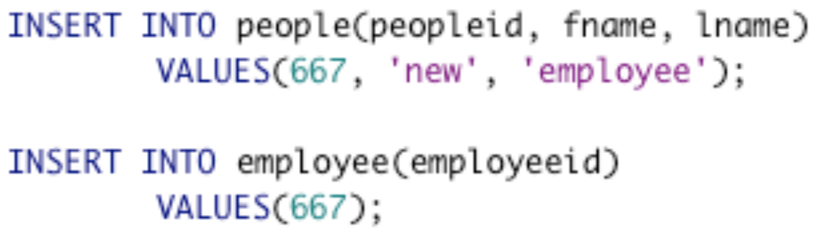
```
create or replace function new_employeeAssign() returns trigger as $new_employeeAssign$
declare
   open_spotID int := (select parkingspot.spotid
                                              from employeepark e full outer join parkingspot on e.spotid = parkingspot.spotid
                                              where (e.spotid <= 2471 and e.spotid >= 2121)
                                              or (e.spotid <= 600 and e.spotid >= 301)
                                              or (e.spotid <= 1900 and e.spotid >= 1601)
                                              or (e.spotid <= 2010 and e.spotid >= 2001)
                                              or (e.spotid <= 2020 and e.spotid >= 2011)
                                              or (e.spotid <= 2120 and e.spotid >= 2021)
                                              or (e.spotid <= 2918 and e.spotid >= 2771)
                                              or (e.spotid <= 3020 and e.spotid >= 2921)
                                              or (e.spotid <= 3220 and e.spotid >= 3021)
                                              or (e.spotid <= 3970 and e.spotid >= 3823) isnull
                                              limit 1);

begin
   insert into employeepark(employeeid, spotid)
               values(new.employeeID ,open_spotID);
   return new;
End;
$new_employeeAssign$ language plpgsql
```

Output is on page with the corresponding trigger.

Trigger

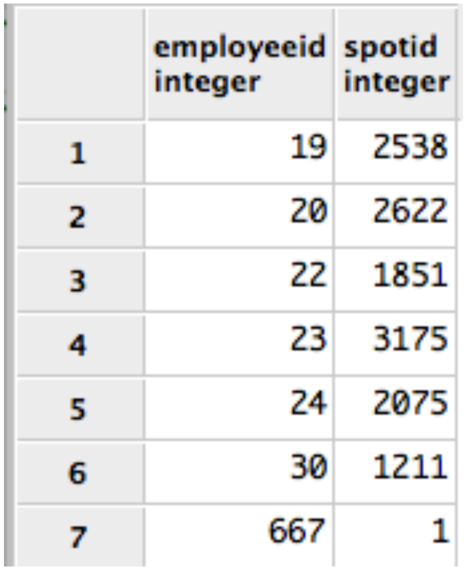
This **trigger** is used to call the stored procedure, new_employeeAssign. This is ran after each new insert into the employee table to automatically assign an employee a parking spot.

```
create trigger new_employeeAssign after insert on employee
   for each row execute procedure new_employeeAssign();
```



```
INSERT INTO people(peopleid, fname, lname)
        VALUES(667, 'new', 'employee');

INSERT INTO employee(employeeid)
        VALUES(667);
```

Employee 667 was added automatically to the table employeeparking and assigned the first available parking spot.



	employeeid integer	spotid integer
1	19	2538
2	20	2622
3	22	1851
4	23	3175
5	24	2075
6	30	1211
7	667	1

Security

Security shows what roles have which privileges within the database. This is useful to avoid potential threats or mistakes queried on the database.

Note this database is not necessarily for the employees or students. This database was created for admins to be able to track Marist's students and employees in regards to where they live on campus, off campus, and where they park, however, it could be useful for employees and students to be able to access their specific tables to see which lots are available to them.

```
-- Admin Role --
Create Role Admin;
Grant All Privileges On All Tables In Schema Public To Admin;

-- Employee --
Create Role Employee;
Grant Select On All Tables In Schema Public to Employee;
Revoke Select on InsideRooms from Employee;

-- Students --
Create Role Students ;
Grant Select On Parkinglot, ParkingSpot, CommuterPark, StudentPark to Students;
```

Conclusion

Throughout the implementation of this database blood, sweat, and tears rained onto my keyboard. Creating each table along with their foreign and primary keys went very smooth, but when it came to stored procedures, there were some difficulties. Having each table relate to one another resulted in tedious queries that would nonsensical to a future database administrator; multiple views – besides the ones that were presented – were used to create relations between tables make these stored procedures possible. The views hid the long and tedious code and allowed a less stressful stored procedure.

Another head-scratcher that was presented was by the security aspect of this database. Determining who the users of this database will be is a major determinant when deciding who has access to what. Ideally, only administrators would have access to this database, keeping track of where students live and park, where employees park, and eventually adding in the entire population of Marist College. For this projects sake, privileges were granted to all, where the students and employees were only allowed to see the lots that the potentially had access too. My final problem was with my stored procedure new_employeeAssign. This stored procedure passed a series of ranges that resembled the potential spots that employees could park in. The stored procedure bypassed these ranges and just returns the overall first available spot, thus assigning the employee to whatever lot that spot may be in. I would like to improve this in the future, assigning the people their correct lot.

Improving Marist College parking is the goal for this database. Where this is just a portion of what is to be expected, future aspirations hope for assigning parking spots to students and employees based on where they live, where they predominately teach, and where their first class of the day is. There are many possible additions to this database and this is just a small portion of a project that could endless, but Rome was not build in one semester.