MARIST SECURITY

Database System

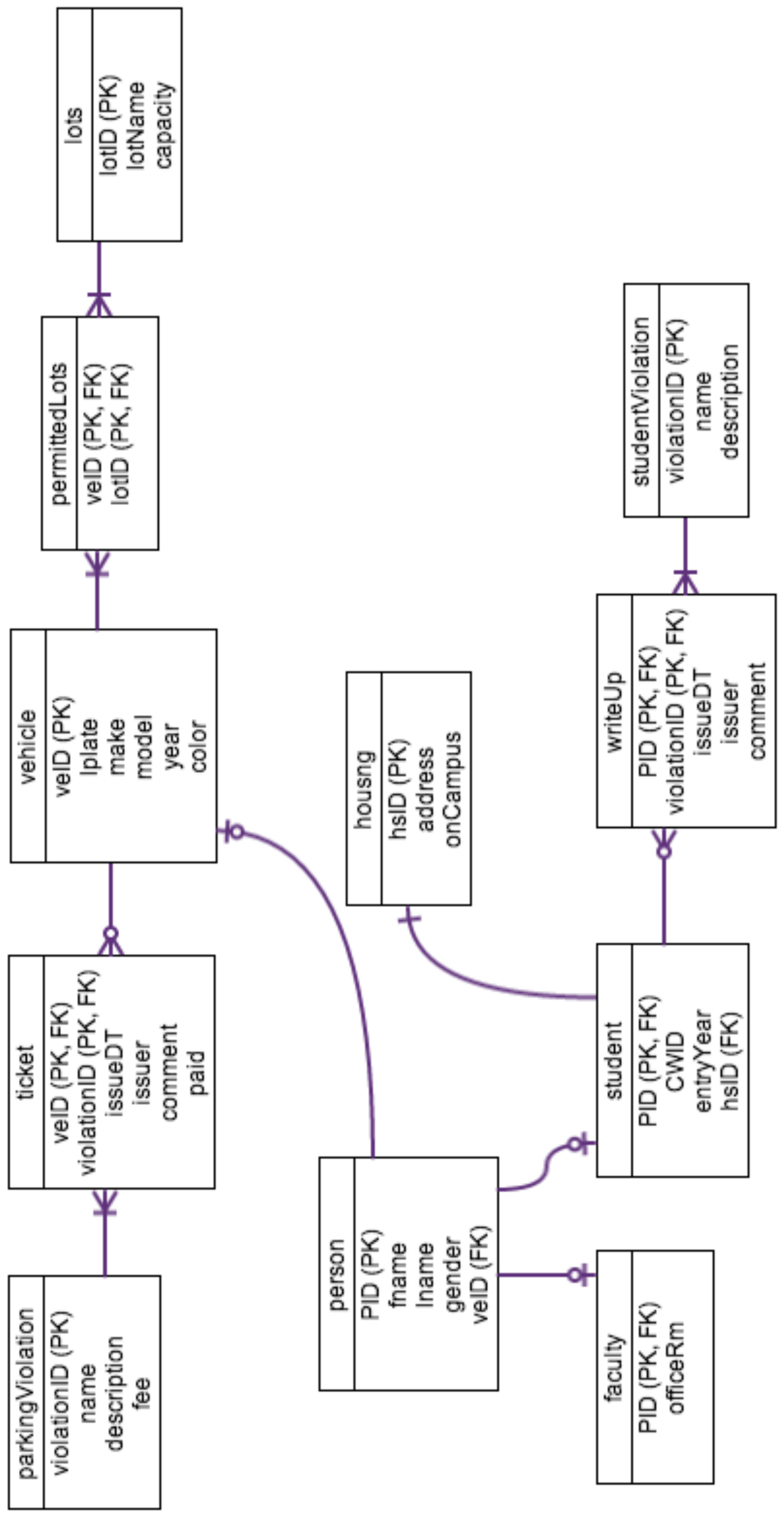
Evan Hopkins

Table of Contents

Cover Page……1
Table of Contents……2
ER Diagram……3
Executive Summary……4
Overview……4
Tables……5 – 15
- Person……5
- Student……6
- Faculty……7
- Vehicle……8
- Permitted Lots……9
- Lots……10
- Ticket……11
- Parking Violation……12
- Write Up……13
- Student Violation……14
- Housing……15

Functional Dependencies……16
Reports……17
Stored Procedures……18
Triggers……19
Views……19
Implementation Notes……20
Security……20
Known Problems……21
Future Enhancements……21

ER Diagram


lots
lotID (PK)
lotName
capacity
permittedLots
veID (PK, FK)
lotID (PK, FK)
vehicle
veID (PK)
lplate
make
model
year
color
ticket
veID (PK, FK)
violationID (PK, FK)
issueDT
issuer
comment
paid
parkingViolation
violationID (PK)
name
description
fee
person
PID (PK)
fname
lname
gender
veID (FK)
housng
hsID (PK)
address
onCampus
studentViolation
violationID (PK)
name
description
writeUp
PID (PK, FK)
violationID (PK, FK)
issueDT
issuer
comment
student
PID (PK, FK)
CWID
entryYear
hsID (FK)
faculty
PID (PK, FK)
officeRm

Executive Summary

This document is a complete outline of the Marist Security database system. The goal of the database is to provide Marist Security with centralized record of all information required to keep students safe. The Entity Relationship Diagram gives an overview of the structure of the database. Following that, a detailed description of each entity is provided. This section includes the reasoning and purpose of the specific entity, as well as the create statements to build the table itself. Also included in this section is an example of fake data to help understand the table. There are some provided example of reports, view, stored procedures, and triggers that are very helpful for using the system.

Overview

The database system can be grouped into four simplified sections. The first section of tables revolves around the people in the database. Every person in the system has an entry in the person table. Each person may then either have a corresponding entry in the student or faculty table.

The vehicle section tracks both parking violations and where cars are permitted to park. People are allowed to have at most one vehicle registered to their name. Through this section, a parking ticket can be linked to a car, which can then be linked to the owner through the person table.

The housing section is one table that tracks where each student is living. It is capable of handling both on and off campus housing options. Housing is only linked to students, and not persons, because security has no concern with where faculty live.

The violations sections tracks when and how students get into trouble with security. It does this by linking any amount of tickets to a specific student. Each ticket then links to a type of violation.

Person

The **person** table holds the base set of information required for every **person** in the database. A person may be extended to either a **faculty** or a **student**, both of which hold more specific information for the given **person**. Note that the **veID** field can potentially be null if a person does not have a vehicle registered.

```
CREATE TABLE person (
        CWID INT PRIMARY KEY,
        fName TEXT NOT NULL,
        lName TEXT NOT NULL,
        birthdate DATE NOT NULL,
        gender BOOL NOT NULL
        veID INT REFERENCES vehicle (veID)
)
```

PID	fName	lName	birthDate	gender	veID
1	Evan	Hopkins	1994-08-03	0	1
2	Michael	Smith	1993-01-01	0	NULL
3	Kelsey	Woldschmidt	1995-01-01	1	NULL
4	Derek	Williams	1992-01-01	0	NULL
5	Sarah	Hopkins	1993-01-01	1	1
6	Alan	Labouseur	1985-01-01	0	2
7	Anne	Matheuse	1980-01-01	1	3
8	Jake	Guy	1991-01-01	0	NULL
9	Sarah	Parker	1993-01-01	1	NULL
10	Kanye	West	1980-01-01	0	4
11	James	Bond	1970-01-01	0	5
12	Lisa	Kent	1992-01-01	1	NULL
13	Karen	Cain	1995-01-01	1	6
14	Jacob	Dewar	1993-01-01	0	7
15	Julia	Vitale	1991-01-01	1	8

Student

The **student** table holds information about a given person that is only required for students. Therefor every **student** has a correspond entry in the **person** table. The relationship is linked through the CWID attribute. For instance, the **entryYear** attribute allows student to be selected based off of their class. The starting year is used (as opposed to graduate year) to account for students that may require an extra semester or are pursuing a graduate program.

```
CREATE TABLE student (
        PID INT PRIMARY KEY,
        CWID INT NOT NULL,
        entryYear INT NOT NULL,
        hsID INT REFERENCES housing (hsID) NOT NULL
)
```

PID	CWID	entryYear	hsID
1	20011111	2012	1
2	20022222	2011	2
3	20033333	2013	2
4	20044444	2010	3
5	20055555	2011	4
8	20066666	2009	2
9	20077777	2011	1
12	20088888	2010	1
13	20099999	2013	2
14	30011111	2011	4
15	30022222	2009	5

Faculty

The **faculty** table holds further information about a **person** that is only required for Marist faculty. Every **faculty** entry has a corresponding entry in the **person** table. This relationship is linked through the **PID** attribute.

```
CREATE TABLE faculty (
        PID INT REFERENCES person (PID) PRIMARY KEY,
        officeRm TEXT
)
```

PID	officeRm
6	HC101
7	HC202
10	FN101
11	DN101

Vehicle

The **vehicle** table holds an entry for every single car that parks on the Marist Campus. The table has attributes such as **make**, **model**, **year**, and **color** so that a car can be identified if security was unable to get a license plate number. Vehicles can be linked to one or more lots in which they are allowed to park. Vehicles can also be ticketed and these tickets are linked to a specific car from the vehicle table. It should be noted that multiple students can be registered to the same car. This was done incase of the occurring, but possible scenario where siblings both attending Marist and use the same car.

```
CREATE TABLE vehicle (
        veID INT PRIMARY KEY,
        make INT NOT NULL,
        model INT NOT NULL,
        year INT NOT NULL,
        color INT NOT NULL,
        lplate TEXT NOT NULL
)
```

veID	make	model	year	color	lplate
1	Mazda	MX-5	1992	Red	LM123
2	Honda	Civic	2007	Blue	EX823
3	Kia	Optima	2005	Black	BR1231
4	Honda	Civic	2000	Silver	OO9283
5	Honda	Accord	2012	Black	P12X99
6	Toyota	Supra	1993	Red	VV7YY1
7	Mazda	RX-8	2008	Silver	12D99
8	Ford	F150	1999	Black	U123

PermittedLots

The PermittedLots table serves as a linking table between a **vehicle** and specific **lots**. A vehicle must have at least one permitted lot, but may also be permitted in multiple lots. It simply links a unique id for a vehicle to a unique id for a lot. If you look at the sample data below, you will notice some duplicate **veID's.** This is how vehicles are permitted in multiple lots. **veID** 2, for example, is allowed to park in three different lots.

```
CREATE TABLE permittedLots (
        veID INT  REFERENCES vehicle (veID) PRIMARY KEY,
        lotID INT  REFERENCES lots (lotID) PRIMARY KEY
)
```

veID	**lotID**
1	4
2	1
3	1
4	6
5	2
6	4
7	4
2	3
3	3
2	2

Lots

The **lots** table contains an entry for every parking lot of the Marist Campus. Each lot contains a **lotName** field, such as 'McCann', and the **capacity** of the lot. The **lotID** uniquely identifies a lot and is used to link a **vehicle** to specific lots.

```
CREATE TABLE lots (
        lotID INT PRIMARY KEY,
        lotName TEXT NOT NULL,
        capacity INT NOT NULL
)
```

lotID	lotName	capacity
1	McCann	200
2	Bryne	20
3	Dyson	75
4	Hoop	200
5	Gartland	200
6	Upper West	100
7	Lower West	100

Ticket

The **ticket** table is used to link an issued ticket to a specific **vehicle**. It works in a similar manner as **permittedLots** in that it links a unique vehicle id to a unite type of **parkingViolation**. The **date**, **issuer**, and **comment** fields allow for some additional information to surround a ticket. In the case where **paid** is NULL, it means there was no fee to be paid.

```
CREATE TABLE ticket (
        veID INT  REFERENCES vehicle (veID) PRIMARY KEY,
        violationID INT  REFERENCES parkingViolation (violationID) PRIMARY KEY
        issueDT TIMESTAMP NOT NULL PRIMARY KEY,
        issuer TEXT NOT NULL,
        paid BOOL,
        comment TEXT
)
```

veID	violationID	paid	issueDT	Issuer	comment
1	1	false	1379248723	J. Campion	NULL
1	3	NULL	1379248923	A. Albrecht	NULL
3	2	true	1379374928	K. Rose	Driver ignored initial warning
4	1	true	1379244729	L. Smith	NULL
7	1	false	1379292837	J. Campion	Headlights were left on

ParkingViolation

The **parkingViolation** table has an entry for every type of parking violation at Marist. Every parking violation must have a **name** and **description** saying what exactly was done to earn the violation. There is an option **fee** attribute if the violation requires the owner to pay a fine.

```
CREATE TABLE parkingViolation (
        violationID INT PRIMARY KEY,
        name TEXT NOT NULL,
        description TEXT NOT NULL,
        fee MONEY
)
```

violationID	name	Description	fee
1	Unauthorized Lot	Car was parked in an unauthorized lot	$20.00
2	Speeding	Car exceeded campus speed limit	$10.00
3	Over Lines	Car parked outside of space lines	NULL

WriteUp

The **writeUp** table contains entry for every time security writes up a **student**. It works in a very similar fashion to the **ticket** table. Each entry is linked to a specific **student** and a specific type of **studentViolation**. Every write up also has a **date**, **time**, **issue**, and optional **comment**.

```
CREATE TABLE writeUp (
        PID INT  REFERENCES student (PID) PRIMARY KEY,
        violationID INT REFERENCES studentViolation (violationID) PRIMARY KEY,
        issueDT TIMESTAMP NOT NULL,
        issuer TEXT NOT NULL,
        comment TEXT
)
```

PID	violationID	issueDT	Issuer	comment
1	1	1372345624	P. Lewis	Student was very cooperative
2	3	1379223454	J. Carl	NULL
3	5	1379234524	D. Lewis	Student attempted to fight security

StudentViolation

The **studentViolation** table has an entry for every possible security offense at Marist. An example of this would be a **student** having alcohol in his room on the dry side of campus. Each entry contains a unique id, a name for the offense, and then a more detailed description of what the offense is.

```
CREATE TABLE studentViolation (
        violationID INT PRIMARY KEY,
        name TEXT NOT NULL,
        description TEXT NOT NULL
)
```

violationID	**Name**	**description**
1	Alcohol Consumption	Underage student consuming alcohol on campus
2	Drug Use	Student using drugs on campus
3	Fire Hazard	Fire hazard found in room
4	Noise Violation	Student making too much noise in room
5	Fighting	Student fought other student(s)

Housing

The **housing** table stores all of the housing options for students at Marist. It supports both on campus and off campus housing. The **address** field stores either the name and room number of the on campus housing or the actual address of off campus housing. The **onCampus** attribute is a flag that is only true when the housing is located on campus.

```
CREATE TABLE studentViolation (
        hsID INT PRIMARY KEY,
        address TEXT NOT NULL,
        onCampus BOOL NOT NULL
)
```

hsID	Address	onCampus
1	Champ 115	True
2	Sheahan 202	True
3	For A6	True
4	Upper West B	True
5	71 Taylor Avenue	False
6	97 Clark Street	False
7	60 Sunset	False

Functional Dependencies

person
PID -> fname, lname, gender, veID

faculty
PID -> officeRm

student
PID -> CWID, entryYear, hsID

writeUp
(PID, violationID) -> issueDT, issuer, comment

studentViolation
violationID -> name, description

housing
hsID -> address, onCampus

vehicle
veID -> lplate, make, model, year, color

ticket
(veID, violationID) -> issueDT, issuer, comment, paid

parkingViolation
violationID -> name, description, fee

permittedLots
NONE

lots
lotID -> lotName, capacity

Reports

Total amount of cars permitted to park in McCann lot

```
SELECT COUNT(veID)
FROM permittedLots, lots
WHERE permittedLots.lotID = lots.lotID
AND lots.name = ‘McCann’
```

All students who have been written up grouped by class

```
SELECT DISTINCT fname, lname
FROM person, student, writeUp
WHERE person.PID = student.PID
AND student.entryYear = 2012
AND student.PID = writeUp.PID
GROUP BY student.entryYear
```

Get the ten most recently issued tickets

```
SELECT parkingViolation.name, ticket.timestamp, ticket.issuer
FROM ticket, parkingViolation
WHERE ticket.veID = parkingViolation.veID
ORDER BY ticket.timestamp
LIMIT 10
```

Stored Procedures

Get car owner given a license plate

This stored procedure is used for when security wants to run plates (look up information about a car through the license plate number). This is done daily by security when each parking lot is checked so it makes sense to use a stored procedure.

```
CREATE FUNCTION carOwner(lplate_lookup TEXT)
RETURNS TABLE(fname TEXT, lname TEXT) AS $$
BEGIN
        SELECT person.fname, person.lname
        FROM person, vehicle
         WHERE person.veID = vehicle.veID
        AND vehicle.lplate = lplate_lookup
END;
$$ LANGUAGE PLPGSQL
```

Call: SELECT carOwner(LM132)
Result: (Evan, Hopkins)

Off Campus House Lookup

This stored procedure is used when security want to see who lives in an off campus house. This is done commonly by security to try and figure out which Marist students are throwing parties in off campus housing. Notice how the addresses are being checked using '~'. This is because security may not enter an address exactly as it appears in the system.

```
CREATE FUNCTION ocHousingCheck(address_lookup TEXT)
RETURNS TABLE(fname TEXT, lname TEXT) AS $$
BEGIN
        SELECT person.fname, person.lname, housing.address
        FROM person, student, housing
         WHERE person.PID = student.PID
        AND student.hsID = housing.hsID
        AND housing.onCampus = false
        AND housing.address ~ address_lookup
END;
$$ LANGUAGE PLPGSQL
```

Call: SELECT ocHousingCheck('71 Taylor Avenue')
Result: (Evan, Hopkins, 71 Taylor avenue)
        (Michael, Smith, 71 Taylor avenue)

Triggers

Completely removing old vehicles from system

When a vehicle is removed from the system (when a student graduates), it needs to be completely removed. When a delete query is run on vehicle to remove a vehicle, this trigger will fire. It simply removes all entries in permittedLots of the car that is going to be deleted.

```
CREATE FUNCTION delCarTrigger()
RETURNS trigger AS $$
BEGIN
        DELETE FROM permittedLots
        WHERE permittedLots.veID NOT EXISTS (
                SELECT vehicle.veID
                FROM vehicle
        )
RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER delCarTrigger
        AFTER DELETE ON vehicle
        FOR EACH ROW
        EXECUTE PROCEDURE delCarTrigger();
```

Views

Off campus Housing

Since both on campus and off campus housing are stored in the same table, it is useful to separate them via a view. This way, a user can query **offCampus** as if it were a table. Since it is a view, no special actions are needed when housing arrangements change each year since it will automatically update each time the view is used.

```
CREATE VIEW offCampus AS
        SELECT *
        FROM housing
        WHERE housing.offCampus = true
```

Student Violation Fee's owed

This view allows the user to see which students still have not paid the fee for violations. This provides the user with an always up to date list of unpaid violations. This is useful for if security is determining whether or not to boot a cars wheel because of outstanding debts.

```
CREATE VIEW feeDebt AS
        SELECT person.fname, person.lname, parkingViolation.fee
        FROM person, ticket, parkingViolation, vehicle
        WHERE ticket.veID = vehicle.veID
        AND vehicle.veID = person.veID
        AND ticket.pvID = parkingViolation.pvID
        AND ticket.paid = false
        GROUP BY person.PID
```

Implementation Notes

- All the timestamp fields (found in **ticket** and **writeUp**) are using a unix timestamp format. This should be noted so that queries can be build to return pretty dates and times instead of a confusing unix timestamp.
- It is common for the amount of vehicles permitted to a lot to exceed the lots capacity. This is because commuters and teachers will not be parked every hour of every day. When implementing the system and adding cars, the lot capacity should be used as a guideline but not a concrete limit.

Security

- The system uses multiple accounts, which allow access to different tables and have different abilities. For instance the parking lot security guards only have the ability to access **parkingViolation**, **ticket**, **vehicle**, **permittedLots**, **lots**, and **person**. A parking security guard would never need to see if a student has ever been written up by their RA, therefore their account should not allow access to that data.
- Off campus housing is tricky because there is no official way for Marist Security to get student's addresses. The students can simply be asked what their address is, but that makes it very easy for them to lie and then be 'off the books'. Since there is not way to verify the accuracy of off campus addresses, it must be taken into consideration that the information could be false.
- The decision was made to use text for the **issuer** attribute in **parkingViolation** and **writeUp**. This originally, these fields contained a **PID**, which referenced the **issuer** in the **person** table. This was done so that there is absolutely no link between the issuer name and their stored information in order to protect the issuer. This way the issuer field can use names like 'J. Davidson' which will be meaningful to the users of the system, but not links any personal information to the issuer.

Known Problems

- There is currently no way to track warnings for both **studentViolations** and **parkingVIolations**. It is common for security and/or an RA to let a student off with a warning if they have not been in trouble before. It is important to track warnings even though they are not official violations.
- There is no easy way to manage students who change cars. Marist only allows students to have one car on campus, therefore the relationship between person and vehicle is one to zero or one.   It is not as simple as removing their old car and adding their new one because parking violations from the old car must still be tied to the student. In the current system, if a student changes their car, all old violations are no longer linked to the student.

Future Enhancements

If more time was given to develop the Marist Security System, here is what would have been added:

- A priority points attribute for each students and a priority point deduction attribute for a student violation. This was not added because it technically related to Marist Security. It would, however, be very convenient to add and easy to implement.
- A special information table to hold information about a student. An example of this would be if a student had a peanut allergy or suicidal thoughts. These kinds of information are very important for keeping students safe, but currently don't have a place in the database. The table would simply link a student unique identifier to a piece of text containing the special information.