Table of Contents

Project Direction Overview

The database that I wish to design will be a database associated with a police department's records management application called Police Source. This application serves as an "in-house" incident record system. It will allow the department to track all activities related to policing in their community. I currently work as a police officer, and this type of application, and the associated database, is something that I interact with daily and is critical for me to effectively do my job. Police Source will be used by the police officers, dispatchers, records clerks, and command staff. In modern policing, it is extremely important that every interaction with the public is properly documented and available for both internal use at the department and available to members of the public due to public records laws. The application will assist the members of the department daily by recording all incidents involving police officers while initiating proactive policing techniques, responding to calls for services, and in any routine patrol. An incident is a broadly used to record anything an officer does while on duty. It can include identity fraud to a domestic violence situation, or even just a traffic complaint. It also pertains to officer-initiated activities such as motor vehicle stops.

Every department in the country needs to have an application and a database that can manage the vast amount of information that is generated daily, not only to have a record of what occurred in the community, but also as a tool that officers can utilize while on patrol. Police Source and the database serve multiple functions. One function is housing vital data that the command staff can analyze to determine statistics, direct specific patrols, and determine areas in the community that are more vulnerable to specific crimes. It also serves as a hub of information that officers can utilize to find out information and demographics about an individual they are interacting with, or interactions between parties at a call for service. Examples of this are the individual's history of interactions with the department to include prior motor vehicle citations, incidents they were involved in, vehicles they own, and personal information to help identify them. It can help the officer make more informed decisions and allow them to adapt to specific threats based on their individual's history.

Police Source tracks activities that police officers perform and the individuals they encounter, but also provides an ability to search based on these activities. The information can be entered into the database by an officer who takes a call for service over the business line or 911, or it could be entered by officers during or after any incident. Police Source would need to provide a user-friendly interface for this to happen efficiently and easily. My focus on this project will be the database behind the application. The database must maintain all the data pertaining to people's demographics such as their names and dates of births. It must also contain history incidents that will have documented incident types such as burglaries, alarms, and motor vehicle stops. Additionally, it should provide dispositions for incidents that summarize the call outcome such as services rendered, citation issued, or arrest made. The database will need to associate people with incidents, people with vehicles, vehicles with incidents, and officers with incidents and possibly other associations that I may realize as I begin to develop it further.

Use Cases and Fields

Use Case 1: *Add an Officer to the Database Use Case*

1. A supervisor logs into the database and creates a new officer.
2. The application asks for the officer's information which the supervisor enters.

3. A new officer is created in the database.

An officer can be associated to incidents or citations.

Field	What it Stores	Why it's needed
Badge_Id	This is the officer's badge number.	This is what an officer uses as a unique identifier on incidents, citations, or arrests.
Off_Firstname	This is first name of the officer.	This is necessary for displaying officers name on screens and adding first name to incidents.
Off_Lastname	This is the last name of the officer.	This is necessary for displaying officers name on screens and adding last name to incidents.
Off_Rank	This is the officer's rank.	This is necessary for displaying officers rank and this distinguishes chain of command between officers. Additionally, can be used as a permission for different level of access in application.
Off_Position	This is the officer's position.	This is necessary for displaying the officer position. Position also determines roles and responsibilities of the officer.
Off_Supervisor	This is the officer's direct supervisor	This is necessary for displaying the officer supervisors. Multiple officers can be assigned to a supervisor and can be used to show chain of command.

Use Case 2*: Add a Person to the Database Use Case*

1. An officer encounters a person during an incident.
2. The officer enters the person's information into the database.
3. A person is created in the database.

A person can be used to associate to incidents, vehicles, or citations. The significant fields are detailed below.

Field	What it Stores	Why it's needed
Per_FirstName	This is first name of the person.	This is necessary for displaying person's name on screens and adding first name to incidents.
Per_LastName	This is the last name of the person.	This is necessary for displaying person's name on screens and adding last name to incidents.

Per_Address	This is the address of the person.	This is necessary for displaying the persons address on screens and associating a person and an address or necessary if officer needs to locate the person.
Per_DOB	This is the date of birth of the person.	This is necessary for displaying the Person's date of birth. This is important for determining the person's age and a way to distinguish individuals.
Per_Phone	This is the phone number of the person.	This is necessary for displaying the person's phone number on screens. This is important as a means to contact the person.
Per_SSN	This is the Social Security Number of the person.	This is necessary for displaying the person's social security number on screens. It is also used as a unique identifier of the person.
Per_OLN	This is the operator's license number of the person.	This is necessary for displaying the person's driver's license number. This is used to associate a person with a citation or to look up the person in a state's motor vehicle registry.

Use Case 3: *Add a Vehicle to the Database Use Case*

1. An officer encounters a vehicle during an incident.
2. The officer enters the vehicle's information into the database.
3. A vehicle is created in the database.

The vehicle would be associated with a person as the owner. A vehicle could also be associated with a person or several people as the operators. Additionally, it can be associated with an incident or multiple incidents. The significant fields are detailed below.

Field	What it Stores	Why it's needed
Veh_Reg	This is the vehicle's license plate number.	This is necessary for displaying the vehicles license plate number on screen. This is a unique identifier for the vehicle.
Veh_VIN	This is the vehicle VIN number.	This is necessary for displaying the vehicles license plate number on screen. This is also a unique identifier for the vehicle.

Veh_Make	This is the vehicle's manufacturer.	This is necessary for displaying vehicle' make on screens. Necessary for citations.
Veh_Model	This is the model of the vehicle.	This is necessary for displaying vehicle' model on screens. Necessary for citations.
Veh_Year	This is the year the vehicle was manufactured.	This is necessary for displaying vehicle' manufacture year on screens. Necessary for citations.
Veh_Color	This is the vehicle's color.	This is necessary for displaying vehicle' color on screens. Necessary for citations.
Veh_Owner	This is name of the owner.	This is necessary for displaying owner's name on screens and for officers to know who the vehicle belongs to. This field associates the vehicle with a person.

Use Case 4: *Add an Incident to the Database Use Case*

1. An incident occurs
2. An officer adds the incident information into to the database.
3. A new incident is created in the database.
4. The officer associates a caller to the incident.
5. The officer associated responding officers to the incident.
6. The officer associates a person or multiple people to the incident.
7. The officer associates person or multiple people to the incident as a participant or participants.
8. The officer selects the type of participant the person is: Caller, Victim, Witness, or Offender.
9. The officer associates a vehicle to the incident.
10. The officer selects the action taken or disposition of the incident.

An incident would be associated with a person or multiple people as the callers, and a person or multiple people involved. It would also be associated with multiple vehicles or no vehicles. The significant fields for an incident are detailed below.

Field	What it Stores	Why it's needed
Inc_Num	This is the incident number.	This is necessary for displaying the incident number on screen. This is used as a unique identifier for the incident.
Inc_Status	This displays what status of the incident.	This is necessary for displaying the incident status on screen. This will can help track the status of the incident. Whether it is open, closed, cancelled, or under investigation.

Inc_Reported	This is how the incident was reported.	This is necessary for displaying how the incident was reported. This can be used to track how incidents are reported.
Inc_Date	This is the date the incident was reported.	This is necessary for displaying the date the incident was reported. Used to track incidents.
Inc_Time	This is the time the incident was reported.	This is necessary for displaying the time the incident was reported. Used to track incidents.
Inc_Category	This is the category that the incident was reported as.	This is necessary for displaying incident category on screens. Necessary to alert officer to what type of incident and to track incidents by categories.
Inc_Location	This is the address where the incident took place.	This is necessary for displaying the incident location on screen. This can associate an incident and an address.
Inc_Officer	This is the primary officer that is assigned the incident.	This is necessary for displaying the reporting officer on screen. This officer is the officer that is primarily responsible for the incident and this associated with the officer by BadgeId.
Inc_Action	This is the action taken on the incident or the disposition of the incident.	This is necessary for displaying the incident disposition on screen. This tracks the outcome of the incident.

Use Case 5: *Add a Citation to the Database Use Case*

1. An incident occurs such as a motor vehicle stop.
2. An officer issues a citation in the incident.
3. The officer selects the type of citation: Cite_Warning, Cite_Fee, Cite_Criminal.
4. The officer adds the citation information to the database.
5. The officer associates a person to the citation as an operator.
6. The officer associated a vehicle to the citation.

A citation must be linked to an incident such as a motor vehicle stop. This citation must include an operator (a person) and a vehicle.

Field	What it Stores	Why it's needed
Cite_Num	This is the Citation number.	This is necessary for displaying the citation number on screen.

		This is used as a unique identifier for the citation and what number is used by the registry on the physical citation.
Cite_type	This displays what type of citation it is.	This is necessary for displaying the type of citation on the screen. There can be multiple types of citations: verbal warnings, written warnings, civil citation, criminal summons, or void.
Cite_Date	This is the date the citation was issued.	This is necessary for displaying the date the citation was issued. This is important for tracking when citations were issued.
Cite_Time	This is the time the citation was issued.	This is necessary for displaying the time the citation was issued. This is important for tracking when citations were issued.
Cite_Loc	This is location where the citation was issued.	This is necessary for displaying the location the citation was issued. This can be used to track where citations are issued.
Cite_Officer	This is the officer that issued the citation.	This is necessary for associating an officer and the citation and displaying it on screen. The citation must also display the officer that issued it. The BadgeId is would be used for this field.
Cite_Operator	This is the operator of the vehicle.	This is necessary to associating a person and the citation and display it on the citation and the screen.
Cite_Vehicle	This is the vehicle that was being operated when the citation was issued.	This is necessary for associating a vehicle and the citation. It also is necessary for displaying the vehicle information on the citation and the screen.
Cite_Incident	This is the incident where the citation was issued.	This is necessary for associating a citation to an incident.
Cite_Amount	This is the fine amount on the citation.	This is necessary for displaying the fine amount listed on the citation. It can be used to track fine amounts generated from citations.

Search Function

An officer would be able to search for another officer, a person, a vehicle, a citation, or an incident by utilizing a search function in the application and any of the fields, or a combination of the fields, as the search criteria.

Structural Database Rules

Use Case 1: *Add an Officer to the Database Use Case*

1. A supervisor logs into the database and creates a new officer.
2. The application asks for the officer's information which the supervisor enters.
3. A new officer is created in the database.

This use case focuses on storing officer data in the database. There is one entity in this use case, Officer, but this use case contains a recursive relationship with the supervisor field. This field creates a relationship within the single Officer entity as an officer may be supervised by another officer. This is a one-to-many (1:M) relationship between Officer and Officer. This would lead to the first rule:

1. An officer may supervise one or more officers; An officer may be supervised by one officer.

I made this structural database rule to show that each officer may have a supervisor, who is another officer. Additionally, an officer may be a supervisor to multiple officers. I ensured that the "officer may supervise" phrase is optional because not every officer will be a supervisor. I also left the "Each officer may be supervised" phrase optional because almost all officers will have a supervisor except the Chief, who is still an officer, will not.

Use Case 2*: Add a Person to the Database Use Case*

1. An officer encounters a person during an incident.
2. The officer enters the person's information into the database.
3. The officer associates a person with an address.
4. A person is created in the database.

Use Case 2 lists five entities, Officer, Address, and Person, Social_Security, and OLN. There is a relationship between the Person entity and the Address entity.

2. A person may have one address. An address may be associated to one or more people.

From the perspective of person, the relationship is optional and singular since every person can only have one address, but not every person may have an address. From the perspective of address, the relationship is plural and optional since not every address may be associated to a person, and multiple people can live at each address.

The use case does not appear to illustrate any relationships between the Officer and Person entities within the database. The use case does not associate an officer to a person in the database beyond an officer adding the person to the database which is not recorded. Therefore, I identified two entities,

Officer and Person, but I do not have enough information for a structural database rule from this use case alone. A person and an incident are associated, but that will be better demonstrated in Use Case 4 through the use of an additional entity, Participant.

Use Case 3: *Add a Vehicle to the Database Use Case*

1. An officer encounters a vehicle during an incident.
2. The officer enters the vehicle's information into the database.
3. A vehicle is created in the database.

Use Case 3 identifies multiple entities: Vehicle, Person, Officer, and Incident. Vehicle has a Person associated to it as an owner. From the first step of the use case, Vehicle must be associated to an incident for an officer to enter it in the database. The use case does not directly relate Officer to Vehicle. An officer encounters the vehicle and enters the information into the database, but the officer entering the information is not recorded in the database as a field under the Vehicle entity. The officer will be associated to an incident, but this will be better illustrated in Use Case 4.

From the Vehicle entity, I've identified two entities that would require structural rules, Incident and Person.

3. Each vehicle is owned by one person; Each person may own one or more vehicles.

Since a vehicle must be owned by a person, this a mandatory relationship. From the perspective of Vehicle, it must be associated to one person making it singular. From the perspective of person, a person may or may not own a vehicle, and the same person could own many vehicles making the relationship optional and plural.

For a vehicle to be added to the database, an officer would have to encounter it during an incident, but a vehicle may also be involved in multiple incidents. Additionally, an incident may involve no vehicles or multiple vehicles. This creates a many-to-many relationship between Vehicle entity and Incident entity. To break this down into two one-to-many relationships I will use an additional entity, Involved_Vic.

4. Each involved_vic is associated to one vehicle; Each vehicle may be associated to one more involved_vic.

From the perspective of Involved_vic, the relationship with vehicle is mandatory and singular since each involved_vic will only be associated to one vehicle and must be associated with a vehicle to be involved in an incident. From the perspective of Vehicle and from use case #3, it does not need to be associated with an incident to be added to the database. It is therefore an optional relationship. Since a vehicle can be involved in multiple incidents as an involved_vic, the relationship is plural.

5. Each involved_vic is involved in one incident; Each incident may involve one or more involved_vic.

From the perspective of Involved_vic, the relationship to incident is mandatory. The relationship is also singular as each involved_vic can only be associated to one incident. From the perspective of incident, an incident could have no involved vehicles, or it may have several in such cases as a multi-car accident making the relationship optional and plural.

Use Case 4: *Add an Incident to the Database Use Case*

1. An incident occurs
2. A new incident is created in the database.
3. The officer associates an address to the incident
4. The officer associates a caller to the incident.
5. The officer associates responding officers to the incident.
6. The officer associates a person or multiple people to the incident.
7. The officer selects the type of participant the person is: Caller, Victim, Witness, or Offender.
8. The officer associates a vehicle to the incident.
9. The officer selects the action taken or disposition of the incident.

Use Case 4 also identifies multiple entities: Incident, Officer, Person, Vehicle, and Address.

Since an incident must have an address, from the perspective of incident the relationship is mandatory and singular since an incident can only have one address. From the perspective of Address, the incident is optional since not every address will have an incident and it is plural since an address may have multiple incidents.

6. Each incident has one address. Each address may have one or more incidents.

Since a few of the entities identified in Use Case 4, Person and Officer, have many-to-many relationships with Incident, I've created two new composite entities to be added: Responder and Participant. I will use these entities to create one-to-many relationships instead of many-to-many relationships between the entities.

The first entity identified is Officer. This entity is related to Incident in two steps in Use Case 4. The first step is when an incident is added to the database. The Inc_Officer is a field that identifies who the primary officer is that manage the incident and will be a foreign key in the Incident table. This officer is primarily responsible for writing a report, booking an arrest, and taking overall responsibility for the incident. This creates a relationship between Officer and Incident in a one-to-many (1:M) relationship.

7. Each officer may manage one or more incidents; Each incident is managed by one officer.

When an officer first gets hired, they will not be primary to any incidents, so from the perspective of Officer, the relationship is optional. Additionally, since an officer can be the primary officer to many incidents, the relationship is plural. From the perspective of Incident, it is a mandatory relationship since an incident must have a primary officer to manage it, and it is a singular relationship since each incident can only have one primary officer.

The Officer entity is also mentioned in step five of Use Case 5: "The officer associates responding officers to the incident." This would be a case of repeating information in the database since at least one officer is already associated with the incident as the primary officer, but the Responder table will include additional necessary information such as the time the officer was associated with the incident. This also creates a many-to-many (M:N) relationship between Officer and Incident because there could be many officers who respond to many different incidents which is why I've decided to create a composite entity.

The composite entity, Responder, creates a one-to-many (1:M) relationship to both Officer and Incident entities and converts the many-to-many relationship into two one-to-many relationships. The Responder entity adds two additional structure rules, rule #5 and rule #6.

8. Each responder responds to one incident; Each incident has one or more responders.

From the perspective of Responder, it is a mandatory and singular relationship. A responder is only listed in the Responder entity if the officer is associated with an incident. The officer must only be associated with one incident per record to create a one-to-many relationship. From the perspective of incident, it is mandatory as every incident must have at least one responder associated to it and it is plural as there could be multiple officers responding to an incident.

9. Each responder is associated to one officer; Each officer may be associated as one or more responders.

From the perspective of responder, it is a mandatory and singular relationship. From the perspective of Officer, it is an optional and plural relationship since a new officer may not have responded to an incident yet, but a veteran officer will have responded to many incidents over their career. Like rule #5, his allows a one-to-many relationship between Responder and Officer to avoid the many-to-many relationship that Officer to Incident would require.

The Participant entity, like the Responder entity is a composite entity. This avoids the many-to-many relationship between person and incident. The next rule relates Participant and Person:

10. Each Participant is a Person; Each Person may be one or more Participants.

From the perspective of Participant, the relationship is mandatory and singular since a participant must be one person. Yet, since a person can participate in multiple incidents, from the perspective of person, it is a plural relationship. It is also an optional relationship since adding a person to the database as illustrated in Use Case 2 does not require the person to be associated with an incident.

Rule #9 relates Participant to Incident:

11. Each participant is involved in one incident; Each incident may involve one or more participants.

Since a participant will not be added to the database unless it is involved in an incident, this is a mandatory and singular relationship from the perspective of participant. From the perspective of incident, an incident may or may not have a participant, or it may have multiple participants. This creates an optional and plural relationship.

After reading through use case 4, I've determined that something that would be useful for Police Source would be to distinguish between types of participants. I've modified the use case for the officer to select the type of participant: Caller, Victim, Witness, or Offender.

This change would enable me to determine an additional structural rule:

12. A participant is a caller, victim, witness, offender, several of these, or none of these.

In this rule, the Participant is the supertype, and caller, victim, witness, and offender are the subtypes. The phrase, “several of these” indicates that it’s an overlapping relationship since a participant can be one of these, several of these, or even all of these. Since a person can be associated to an incident as a participant, but may not be one of these four subclasses, I’ve added the phrase, “or none of these” to indicate that the relationship is partially complete. The phrases I would use for this relationship in the UML ERD is “{optional, and}.”

Use Case 5: *Add a Citation to the Database Use Case*

1. An incident occurs such as a motor vehicle stop.
2. An officer issues a citation in the incident.
3. The officer selects the type of citation: Cite_Warning, Cite_Fee, Cite_Criminal.
4. The officer adds the citation information to the database.
5. The officer associates a person to the citation as an operator.
6. The officer associated a vehicle to the citation.

Use Case 5 involves the entities already identified in other use cases, Person, Officer, Incident, and Vehicle. A citation creates a one-to-many relationship between these entities. This leads to a structural rule for each of these associations.

13. Each citation cites one person; Each person may have one or more citations.

From the perspective of citation, there is only one person being cited, the operator, and a citation must have an operator for it to be valid. It is a mandatory and singular relationship. From the perspective of Person, a person may or may not have ever been cited, but a person could have had multiple citations. This is an optional and plural relationship.

14. Each citation is issued by an officer; Each officer may issue one or more citations.

From the perspective of citation, it must be issued by an officer making it a mandatory relationship and since a citation cannot be issued by multiple officers, it is singular. From the perspective of Officer, a new officer may have never issued a citation making the relationship optional, but a veteran officer may have issued thousands of citations make it a plural relationship.

15. Each citation is issued in one incident; Each incident may have multiple citations.

A singular citation cannot be issued multiple times so it would only occur in one incident, and it can only be issued in an incident leading to a mandatory and singular relationship. From the perspective of incident, an incident may not have a citation, but multiple citation could be issued in one incident in such situations as a multiple vehicle crash or with enough infractions to warrant multiple citations. This means it is an optional and plural relationship.

16. Each citation is associated one vehicle; Each vehicle may be associated to one or more citations.

A citation only lists one vehicle per citation, making it a mandatory and singular relationship from the perspective of citation. From the perspective of vehicle, the relationship is optional and plural since a

vehicle may never be associated with a citation, but the same vehicle could be associated with multiple citations just a single person could be in rule #9.

When thinking about citations, I realized that there are several different types. A citation can be a written warning, money citation, or a criminal citation. These different types would be useful for Police Source to distinguish between. I've modified the use case for the officer to select the type of citation: Cite_Warning, Cite_Fee, Cite_Criminal.

17. A citation is a cite_warning, cite_fee, or cite_criminal.

This database has only three kinds of citations. The relationship is disjoint and complete. I did not add any additional phrases to the rule because there are no other kinds of citations available, and a citation must be one of these three types. The UML phrase I would use for this relationship in the ERD is, "{mandatory, or}."

Every change of the cite_criminal status attribute will be recorded in the Status_Change history table. Its relationship with cite_criminal is:

18. **Each cite_criminal may have one or more status_changes; each status change is associated to one cite_criminal.**

The relationship from the perspective of cite_criminal is plural and optional since a cite_criminal may have none or many status_changes. From the perspective of status_change the relationship is singular and mandatory since each status_change must associate to only one cite_criminal.

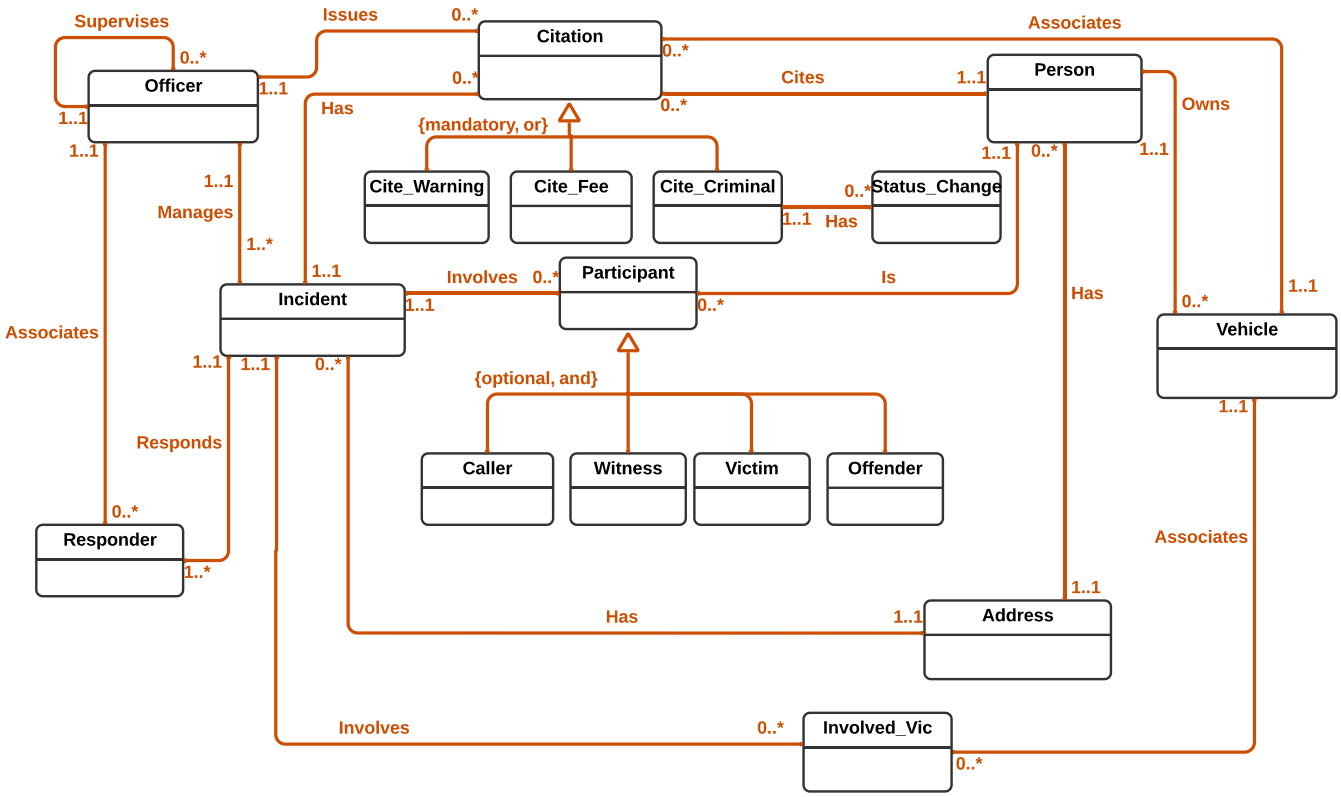
By going through the use cases, I have identified 17 entities:

1. Officer
2. Person
3. Vehicle
4. Involved_vic
5. Incident
6. Participant
7. Offender
8. Victim
9. Witness
10. Caller
11. Responder
12. Citation
13. Cite_criminal
14. Cite_fee
15. Cite_warning
16. Address
17. Status_Change

I have also identified 18 Structural Rules:

1. An officer may supervise one or more officers; An officer may be supervised by one officer.
2. Each vehicle is owned by one person; Each person may own one or more vehicles.
3. Each involved_vic is associated to one vehicle; Each vehicle may be associated to one more involved_vic.
4. Each involved_vic is involved in one incident; Each incident may involve one or more involved_vic.
5. Each officer may manage one or more incidents; Each incident is managed by one officer.
6. Each responder responds to one incident; Each incident has one or more responders.
7. Each responder is associated to one officer; Each officer may be associated as one or more responders.
8. Each Participant is a Person; Each Person may be one or more Participants.
9. Each participant is involved in one incident; Each incident may involve one or more participants.
10. A participant is a caller, victim, witness, offender, several of these, or none of these.
11. Each citation cites one person; Each person may have one or more citations.
12. Each citation is issued by an officer; Each officer may issue one or more citations.
13. Each citation is issued in one incident; Each incident may have multiple citations.
14. Each citation is associated one vehicle; Each vehicle may be associated to one or more citations.
15. A citation is a cite_warning, cite_fee, or cite_criminal.
16. Each Incident has an address. An address may have one or more incidents.
17. Each Person has an address. An address may have one or more people.
18. **Each cite_criminal may have one or more status_changes; each status change is associated to one cite_criminal.**

Conceptual Entity-Relationship Diagram



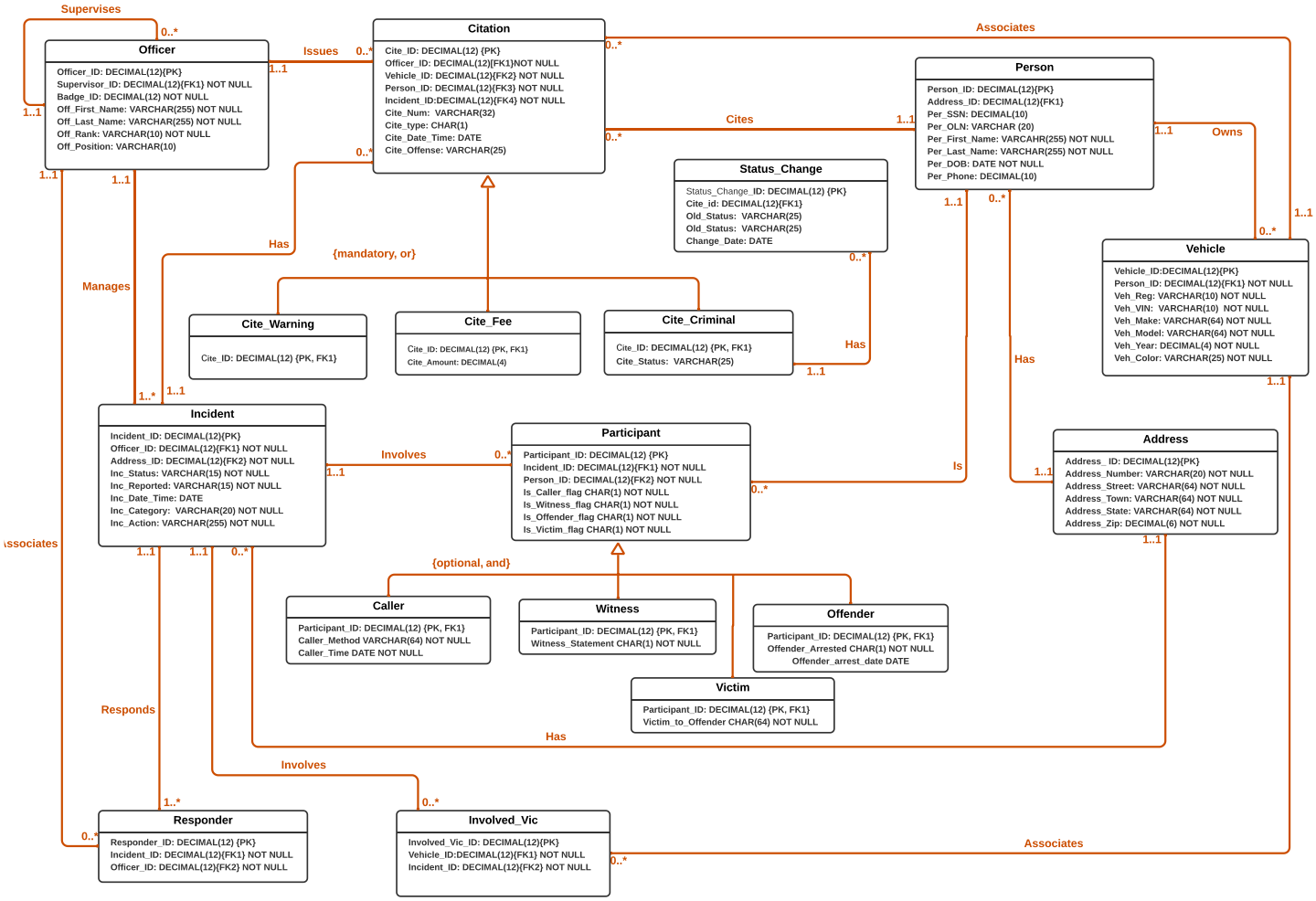
Full DBMS Physical ERD

Table	Attribute	Datatype	Reasoning
Officer	Off_Badge_Id	DECIMAL(6)	This is the Officer badge number. Every officer has a badge number, it is a unique number. I've allowed up to 6 digits in case the system is used by a large department.
Officer	Off_Firstname	VARCHAR(255)	This is first name of the officer, up to 255 characters for the first name.
Officer	Off_Lastname	VARCHAR(255)	This is the last name of the officer, up to 255 characters for the last name.
Officer	Off_Rank	VARCHAR(10)	This is the officer's rank, up to 10 characters for the rank.
Officer	Off_Position	VARCHAR(10)	This is the officer's position, up to 10 characters for the position.
Officer	Off_Supervisor	DECIMAL(12)	Foreign key (recursive) references officer (officer_id). This is the officer's direct supervisor.
Person	Social_ID	DECIMAL(10)	Foreign Key references Social_Security table. It is the person's social security number.
Person	OLN_Id	DECIMAL(12)	Foreign Key references OLN table. It is the person's operator license number.

Person	Per_FirstName	VARCHAR(255)	This is first name of the person, up to 255 characters for the first name.
Person	Per_LastName	VARCHAR(255)	This is the last name of the person, up to 255 characters for the first name.
Person	Address_Id	DECIMAL(12)	Foreign Key to Address. This is the address of the person.
Person	Per_DOB	DATE	This is the date of birth of the person.
Person	Per_Phone	DECIMAL(10)	This is the phone number of the person; a phone number consists of 10 digits.
Person	Per_SSN	DECIMAL(10)	This is the Social Security Number of the person, a social security number consists of 10 digits.
Person	Per_OLN	VARCHAR(20)	This is the operator's license number of the person, it allows up to 20 characters as a license may contain a mix of letters and numbers depending on the state of issue.
Participant	Is_Caller_flag	Char(1)	This is a flag to indicate whether a participant is a caller.
Participant	Is_Victim_flag	Char(1)	This is a flag to indicate whether a participant is a victim.
Participant	Is_Witness_flag	Char(1)	This is a flag to indicate whether a participant is a witness
Participant	Is_Offender_flag	Char(1)	This is a flag to indicate whether a participant is an offender.
Vehicle	Veh_Reg	VARCHAR(10)	This is the vehicle's license plate number. It allows up to 15 characters as it may be a combination of letters and numbers and vary on length depending on the state of issue.
Vehicle	Veh_VIN	VARCHAR(17)	This is the vehicle VIN number. A VIN number consists of a mix of letters and digits but it is exactly 17 characters long.
Vehicle	Veh_Make	VARCHAR(64)	This is the vehicle's manufacturer. It allows 64 characters in case of a long manufacture name.
Vehicle	Veh_Model	VARCHAR(64)	This is the model of the vehicle. It allows 64 characters in case of a long model name.
Vehicle	Veh_Year	DECIMAL(4)	This is the year the vehicle was manufactured. It is 4 digits to represent the year manufactured.
Vehicle	Veh_Color	VARCHAR(25)	This is the vehicle's color. It allows 25 characters for vehicle color.

Vehicle	Person_ID	DECIMAL(12)	This is name of the owner. It is a foreign key to the person table.
Involved_Vic	Vehicle_ID	DECIMAL(12)	Foreign Key references Vehicle.
Involved_Vic	Incident_ID	DECIMAL(12)	Foreign Key references Incident.
Incident	Inc_Status	VARCHAR(15)	This displays what status of the incident. It allows up to a 15 character status.
Incident	Inc_Reported	VARCHAR(15)	This is how the incident was reported. It allows up to a 15 character entry. It will be a drop down in the application to either 911, business line, proactive or possibly other options.
Incident	Inc_Date_Time	DATE	This is the date and time the incident was reported.
Incident	Inc_Category	VARCHAR(20)	This is the category that the incident was reported as, it allows up to 20 characters for category types.
Incident	Address_Id	DECIMAL(12)	Foreign Key references Address. This is the address where the incident took place.
Incident	Officer_Id	DECIMAL(12)	Foreign Key references Officer. This is the primary officer that is assigned the incident.
Incident	Inc_Action	VARCHAR(255)	This is the action taken on the incident or the disposition of the incident. The application will have predefined drop-down list of possible actions taken. It will allow up to 255 characters.
Responder	Officer_ID	DECIMAL(12)	Foreign Key references Officer. This a responding officer.
Responder	Incident_ID	DECIMAL(12)	Foreign Key references Incident, this is the incident the responding officer responds to.
Citation	Cite_Num	VARCHAR(32)	This is the Citation number. Citation numbers are a combination of characters and digits. It will allow up to 32 characters for the citation number.
Citation	Cite_type	CHAR(1)	There are three types of citations. This is an indicator to distinguish which subtype it is.
Citation	Cite_Date_Time	DATE	This is the date and time the citation was issued.
Citation	Address_Id	DECIMAL(12)	Foreign Key references Address. This is the address where the citation was issued.
Citation	Officer_Id	DECIMAL(12)	Foreign Key references Officer. This is the officer that issued the citation.
Citation	Person_Id	DECIMAL(12)	Foreign Key references Person. This is the operator of the vehicle.
Citation	Vehicle_Id	DECIMAL(12)	Foreign Key references Vehicle. This is the vehicle that was being operated when the citation was issued.

Citation	Incident_Id	DECIMAL(12)	Foreign Key references Incident. This is the incident where the citation was issued.
Citation	Cite_Offense	VARCHAR(25)	This is the offense the citation was issued for. It allows up to 25 characters.
Cite_fine	Cite_Amount	DECIMAL(4)	This is the fine amount on the citation. A citation fine amount does not have decimals, and it would never be bigger than $9999.
Cite_Criminal	Cite_Status	VARCHAR(25)	This is the status of the criminal case on the citation, it allows up to 25 characters.
Address	Address_Number	VARCHAR(20)	This is the address number, as addresses could contain letters it allows up to a combination of 20 characters and digits.
Address	Address_Street	VARCHAR(64)	This is the street name. As some street may have long names it allows up to 64 characters.
Address	Address_Town	VARCHAR(64)	This is the town name. As some towns may have long names it allows up to 64 characters.
Address	Address_State	VARCHAR(64)	This is the state name
Address	Address_Zipcode	DECIMAL(6)	This is the state 6 digit Zip code
Offender	Offender_Arrested	CHAR(1)	This indicates if the offender was arrested.
Offender	Offender_Arrest_Date	DATE	This is the date the offender was arrested.
Witness	Witness_Statement	CHAR(1)	This indicates whether the witness completed a voluntary statement
Victim	Victim_To_Offender	VARCHAR(64)	This is the relation of the victim to offender. It allows 64 characters to describe such as mother, friend, brother, none, etc.
Caller	Call_method	VARCHAR(64)	This is the method the caller used such as 911, business line, text. It allows 64 characters to describe.
Caller	Call_time	DATE	This is the time of the call
Status_Change	**Old_Status**	**VARCHAR(25)**	**This is the old status before updating**
Status_Change	**New_Status**	**VARCHAR(25)**	**This is the new status after updating**
Status_Change	**Cite_ID**	**DECIMAL(12)**	**Foreign Key references cite_criminal. This is the citation that was updated.**
Status_Change	**Change_Date**	**DATE**	**This is the date the status was changed**



Stored Procedure Execution and Explanations

First two stored procedures are designed from Use Case 2:

Use Case 2*: Add a Person to the Database Use Case*

1. An officer encounters a person during an incident.
2. The officer enters the person's information into the database.
3. The officer associates a person with an address.
4. A person is created in the database.

The first stored procedure I created is inputting an address into the database.

```
173 CREATE OR REPLACE PROCEDURE add_address (
174     add_number VARCHAR,
175     add_street VARCHAR,
176     add_town VARCHAR,
177     add_state VARCHAR,
178     add_zip DECIMAL)
179 IS
180 BEGIN
181     INSERT INTO Address (address_id, address_number, address_street, address_town, address_state, address_zip)
182     VALUES (address_seq.nextval, add_number, add_street, add_town, add_state, add_zip);
183 END;
184 /
```

Script Output | Task completed in 0.052 seconds

```
Procedure ADD_ADDRESS compiled
```

I created a stored procedure named “add_address”. I created parameters that correspond to the fields in the address table to allow these to be inputted. Inside the stored procedure there is an insert statement to insert the data into the address table.

Here is a screenshot of my stored procedure execution:

```
186 BEGIN
187   ADD_ADDRESS (123, 'East Main Street', 'Norton', 'Massachusetts', 12345);
188   ADD_ADDRESS (1, 'West Main Street', 'Norton', 'Massachusetts', 12345);
189   ADD_ADDRESS (2, 'Leonard Street', 'Norton', 'Massachusetts', 12345);
190   ADD_ADDRESS (3, 'Plain Street', 'Norton', 'Massachusetts', 12345);
191   ADD_ADDRESS (5, 'Burt', 'Norton', 'Massachusetts', 12345);
192   ADD_ADDRESS (10, 'Newland Street', 'Norton', 'Massachusetts', 12345);
193   ADD_ADDRESS (11, 'Wilbur Street', 'Norton', 'Massachusetts', 12345);
194   ADD_ADDRESS (22, 'Bay Road', 'Norton', 'Massachusetts', 12345);
195   ADD_ADDRESS (5, 'Johnson Street', 'Norton', 'Massachusetts', 12345);
196   ADD_ADDRESS (7, 'Tipping Place', 'Norton', 'Massachusetts', 12345);
197   END;
198   /
199
```

Script Output | Query Result | Task completed in 0.191 seconds

```
PL/SQL procedure successfully completed.
```

And the resulting table:

```
200 SELECT *
201 FROM address;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.022 seconds

	ADDRESS_ID	ADDRESS_NUMBER	ADDRESS_STREET	ADDRESS_TOWN	ADDRESS_STATE	ADDRESS_ZIP
1	1	123	East Main Street	Norton	Massachusetts	12345
2	2	1	West Main Street	Norton	Massachusetts	12345
3	3	2	Leonard Street	Norton	Massachusetts	12345
4	4	3	Plain Street	Norton	Massachusetts	12345
5	5	5	Burt	Norton	Massachusetts	12345
6	6	10	Newland Street	Norton	Massachusetts	12345
7	7	11	Wilbur Street	Norton	Massachusetts	12345
8	8	22	Bay Road	Norton	Massachusetts	12345
9	9	5	Johnson Street	Norton	Massachusetts	12345
10	10	7	Tipping Place	Norton	Massachusetts	12345

I've also created a trigger to ensure the zip code is 5 digits:

```
272 CREATE OR REPLACE TRIGGER address_Zip_trg
273 BEFORE UPDATE OR INSERT ON Address
274 FOR EACH ROW
275 BEGIN
276     IF (LENGTH(:NEW.address_zip) <> 5) THEN
277     RAISE_APPLICATION_ERROR(-20001, 'The Address Zip Code must be 5 digits.');
278     END IF;
279 END;
280 /
```

The next stored procedure adds a person to the database and associates the person to an address. It is named "add_person". This procedure also adds the address to the database if it does not yet exist. I created parameters that correspond to the fields in the person and address tables to allow these to be inputted. The procedure creates two variables, "person_address_id_number" and "person_address_id." The person_address_id_number holds a value for the number of address_id's in the address table that correspond to the address parameter's given. If this number is 0, then an insert statement adds the new address to the address table. If the number is greater than 0, it does nothing. This allows multiple people to be associated to an address without duplicating the address when a new person is added to the database. The next step in the procedure is to add the new person to the person table with the parameters provided. The procedure also uses the parameters of the address in a subquery to associate the person with an address from the address table.

```
186 CREATE OR REPLACE PROCEDURE add_person (
187     P_ssn DECIMAL,
188     p_OLN VARCHAR,
189     p_first_name VARCHAR,
190     p_last_name VARCHAR,
191     p_DOB DATE,
192     P_phone DECIMAL,
193     add_number VARCHAR,
194     add_street VARCHAR,
195     add_town VARCHAR,
196     add_state VARCHAR,
197     add_zip DECIMAL)
198 IS
199     Person_address_id_number DECIMAL(12);
200     Person_address_id DECIMAL(12);
201 BEGIN
202     SELECT count(address_id)
203     INTO Person_address_id_number
204     FROM address
205     WHERE address_number = add_number AND address_street = add_street AND address_zip = add_zip;
206
207     IF Person_address_id_number = 0 THEN
208     INSERT INTO Address (address_id, address_number, address_street, address_town, address_state, address_zip)
209     VALUES (address_seq.nextval, add_number, add_street, add_town, add_state, add_zip);
210     END IF;
211
212     SELECT address_id
213     INTO Person_address_id
214     FROM address
215     WHERE address_number = add_number AND address_street = add_street AND address_zip = add_zip;
216
217     INSERT INTO Person (person_id, address_id, per_SSN, per_oln, per_first_name, per_last_name, per_DOB, per_phone)
218     VALUES (person_seq.nextval, person_address_id, p_ssn, p_oln, p_first_name, p_last_name, p_DOB, p_phone);
219 END;
220 /
```

Script Output

Task completed in 0.088 seconds

```
Procedure ADD_PERSON compiled
```

Here is a screenshot of my stored procedure execution:

```
BEGIN
ADD_Person (1111111110, 'S12345678', 'John', 'Smith', CAST('1-JAN-1980' AS DATE),
5082341234, 123, 'East Main Street', 'Norton', 'Massachusetts', '12345');
ADD_Person (1234567890, 'S12225678', 'Helen', 'Smith', CAST('1-MAR-1982' AS DATE),
5085557634, 1, 'West Main Street', 'Norton', 'Massachusetts', 12345);
ADD_Person (1117771110, null, 'Grace', 'Smith', CAST('11-JAN-2000' AS DATE),
5085575434, 2, 'Leonard Street', 'Norton', 'Massachusetts', 12345);
ADD_Person (1114444110, 'S16665678', 'Keith', 'Michaels', CAST('1-DEC-2000' AS DATE),
5085445234, 3, 'Plain Street', 'Norton', 'Massachusetts', 12345);
ADD_Person (1133331110, 'S12344478', 'Robert', 'Jones', CAST('4-NOV-2000' AS DATE),
5085976234, 5, 'Burt', 'Norton', 'Massachusetts', 12345);
ADD_Person (1122221110, 'S12377778', 'Mary', 'Cota', CAST('17-JUN-2000' AS DATE),
5085557774, 10, 'Newland Street', 'Norton', 'Massachusetts', 12345);
ADD_Person (1999911110, 'S17775678', 'Jake', 'Boxer', CAST('4-MAY-2000' AS DATE),
5085544664, 11, 'Wilbur Street', 'Norton', 'Massachusetts', 12345);
ADD_Person (1994444110, 'S15785678', 'Thomas', 'Brigg', CAST('22-AUG-2000' AS DATE),
5085235334, 22, 'Bay Road', 'Norton', 'Massachusetts', 12345);
ADD_Person (1223911110, 'S12777678', 'Larry', 'Summers', CAST('11-JUL-2000' AS DATE),
5085523434, 5, 'Johnson Street', 'Norton', 'Massachusetts', 12345);
ADD_Person (1666661110, 'S14776678', 'John', 'Hail', CAST('21-JAN-2000' AS DATE),
5081551234, 7, 'Tipping Place', 'Norton', 'Massachusetts', 12345);
END;
/
```

Script Output

Task completed in 0.179 seconds

```
PL/SQL procedure successfully completed.
```

Here is the resulting table:

```
SELECT *
FROM Person;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.021 seconds

	PERSON_ID	PER_SSN	PER_OLN	ADDRESS_ID	PER_FIRST_NAME	PER_LAST_NAME	PER_DOB	PER_PHONE
1	1	1111111110	S12345678	1	John	Smith	01-JAN-80	5082341234
2	2	1234567890	S12225678	2	Helen	Smith	01-MAR-82	5085557634
3	3	1117771110	(null)	3	Grace	Smith	11-JAN-00	5085575434
4	4	1114444110	S16665678	4	Keith	Michaels	01-DEC-00	5085445234
5	5	1133331110	S12344478	5	Robert	Jones	04-NOV-00	5085976234
6	6	1122221110	S12377778	6	Mary	Cota	17-JUN-00	5085557774
7	7	1999911110	S17775678	7	Jake	Boxer	04-MAY-00	5085544664
8	8	1994444110	S15785678	8	Thomas	Brigg	22-AUG-00	5085235334
9	9	1223911110	S12777678	9	Larry	Summers	11-JUL-00	5085523434
10	10	1666661110	S14776678	10	John	Hail	21-JAN-00	5081551234

I've added 4 triggers for error checking when inputting a person. The first trigger named "person_dog_trg" ensures the person's date of birth is after 1-JAN-1900. This is to ensure that a person is not added to the database with a date of birth that would put them over 120 years old. The second trigger named "person_SSN_trg" ensures the person's social security number is 10 digits. The third trigger named "person_phone_trg" ensures the person's phone number is 10 digits as well. The last

trigger named “check_SSN_redundancy” ensures that no two people are added with the same social security number.

```
242 CREATE OR REPLACE TRIGGER person_DOB_trg
243 BEFORE UPDATE OR INSERT ON person
244 FOR EACH ROW
245 BEGIN
246     IF :NEW.per_DOB < CAST( '1-JAN-1900' AS DATE) THEN
247     RAISE_APPLICATION_ERROR(-20001, 'The date of birth must be after January 1, 1900');
248     END IF;
249 END;
250 /
251
252 CREATE OR REPLACE TRIGGER person_SSN_trg
253 BEFORE UPDATE OR INSERT ON person
254 FOR EACH ROW
255 BEGIN
256     IF (LENGTH(:NEW.per_SSN) <> 10) THEN
257     RAISE_APPLICATION_ERROR(-20001, 'The Social Security number must be 10 digits');
258     END IF;
259 END;
260 /
261
262 CREATE OR REPLACE TRIGGER person_phone_trg
263 BEFORE UPDATE OR INSERT ON person
264 FOR EACH ROW
265 BEGIN
266     IF (LENGTH(:NEW.per_phone) <> 10) THEN
267     RAISE_APPLICATION_ERROR(-20001, 'The phone number must be 10 digits');
268     END IF;
269 END;
270 /
271
```

```
282 CREATE OR REPLACE TRIGGER check_SSN_redundancy
283 BEFORE INSERT OR UPDATE ON Person
284 FOR EACH ROW
285 DECLARE
286     person_ssn_number DECIMAL(1);
287 BEGIN
288     SELECT COUNT(per_ssn)
289     INTO person_SSN_Number
290     FROM Person
291     WHERE :NEW.per_SSN = person.per_ssn;
292
293     IF person_SSN_Number > 0
294     THEN
295        RAISE_APPLICATION_ERROR (-20001, 'This Social Security Number is already in use.');
296     END IF;
297 END;
298 /
```

The third stored procedure I’ve created is built from Use Case 3:

Use Case 3: *Add a Vehicle to the Database Use Case*

1. An officer encounters a vehicle during an incident.
2. The officer enters the vehicle's information into the database.
3. A vehicle is created in the database.

I created a stored procedure named "add_officer". I created parameters that correspond to the fields in the officer table to allow these to be inputted. Inside the stored procedure there is a variable named "off_super_id" which holds the officer_id of the new officer's supervisor. I use this variable in the insert statement to insert the officer_id associated with the supervisor into the new officer record. It also inserts the rest of the parameter data.

```
224 CREATE OR REPLACE PROCEDURE add_officer (
225     O_Badge_id DECIMAL,
226     O_First_Name VARCHAR,
227     O_Last_Name VARCHAR,
228     O_Rank VARCHAR,
229     O_Position VARCHAR,
230     s_badge_id DECIMAL)
231 IS
232     off_super_id DECIMAL(12);
233 BEGIN
234     SELECT officer_id
235     INTO off_super_id
236     FROM officer
237     WHERE s_badge_id = badge_id;
238
239     INSERT INTO Officer (Officer_ID, Supervisor_id, Badge_id, Off_First_Name, Off_Last_Name, Off_Rank, Off_Position)
240     VALUES (officer_seq.nextval, off_super_id, o_Badge_id, o_First_Name, O_Last_Name, O_Rank, O_Position);
241 END;
242 /
```

Script Output × Query Result ×

Task completed in 0.108 seconds

```
Procedure ADD_OFFICER compiled
```

Here is a screenshot of my stored procedure execution, I found that I needed to insert the Chief first, separately from the add_officer procedure as the Chief has no supervisor but is the top of the supervisory pyramid for all other officers:

```
347  INSERT INTO officer
348  VALUES (officer_seq.nextval, null, 134, 'Sean', 'Worrall', 'Chief', 'Administration');
349
350  BEGIN
351  Add_Officer (133, 'Michael', 'Jacobs', 'Lieutenant', 'Patrol', 134);
352  Add_Officer (123, 'Jake', 'Johnson', 'Sergeant', 'Patrol', 133);
353  Add_Officer (333, 'Patrick', 'Booher', 'Patrolman', 'Patrol', 123);
354  Add_Officer (130, 'Robert', 'Goodwin', 'Patrolman', 'Patrol', 123);
355  Add_Officer (143, 'Ashley', 'Kennedy', 'Patrolman', 'Patrol', 123);
356  Add_Officer (190, 'Brian', 'Clark', 'Sergeant', 'Dectective', 134);
357  Add_Officer (173, 'Nancy', 'Leeland', 'Patrolman', 'Dectective', 190);
358  Add_Officer (564, 'James', 'Murphy', 'Sergeant', 'Patrol', 134);
359  Add_Officer (190, 'Janna', 'Perez', 'Patolman', 'Patrol', 564);
360  END;
361  /
```

Script Output | Query Result

Task completed in 0.159 seconds

```
Procedure ADD_OFFICER compiled


1 row inserted.


PL/SQL procedure successfully completed.
```

The resulting table:

```
369  SELECT *
370  FROM officer;
371
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.011 seconds

	OFFICER_ID	SUPERVISOR_ID	BADGE_ID	OFF_FIRST_NAME	OFF_LAST_NAME	OFF_RANK	OFF_POSITION
1	1	(null)	134	Sean	Worrall	Chief	Administration
2	2	1	133	Michael	Jacobs	Lieutenant	Patrol
3	3	2	123	Jake	Johnson	Sergeant	Patrol
4	4	3	333	Patrick	Booher	Patrolman	Patrol
5	5	3	130	Robert	Goodwin	Patrolman	Patrol
6	6	3	143	Ashley	Kennedy	Patrolman	Patrol
7	7	1	190	Brian	Clark	Sergeant	Dectective
8	8	7	173	Nancy	Leeland	Patrolman	Dectective
9	9	1	564	James	Murphy	Sergeant	Patrol
10	10	9	190	Janna	Perez	Patolman	Patrol

Question Identification and Explanations

First two stored procedures are designed from Use Case 2:

Use Case 2*: Add a Person to the Database Use Case*

5. An officer encounters a person during an incident.
6. The officer enters the person's information into the database.
7. The officer associates a person with an address.
8. A person is created in the database.

The first stored procedure I created is inputting an address into the database.

```
CREATE OR REPLACE PROCEDURE add_address (
    add_number VARCHAR,
    add_street VARCHAR,
    add_town VARCHAR,
    add_state VARCHAR,
    add_zip DECIMAL)
IS
BEGIN
    INSERT INTO Address (address_id, address_number, address_street, address_town, address_state, address_zip)
    VALUES (address_seq.nextval, add_number, add_street, add_town, add_state, add_zip);
END;
/
```

Script Output | Task completed in 0.052 seconds

```
Procedure ADD_ADDRESS compiled
```

I created a stored procedure named "add_address". I created parameters that correspond to the fields in the address table to allow these to be inputted. Inside the stored procedure there is an insert statement to insert the data into the address table.

Here is a screenshot of my stored procedure execution:

```
BEGIN
ADD_ADDRESS (123, 'East Main Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (1, 'West Main Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (2, 'Leonard Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (3, 'Plain Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (5, 'Burt', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (10, 'Newland Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (11, 'Wilbur Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (22, 'Bay Road', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (5, 'Johnson Street', 'Norton', 'Massachusetts', 12345);
ADD_ADDRESS (7, 'Tipping Place', 'Norton', 'Massachusetts', 12345);
END;
/
```

Script Output | Query Result | Task completed in 0.191 seconds

```
PL/SQL procedure successfully completed.
```

And the resulting table:

```
200 SELECT *
201 FROM address;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.022 seconds

	ADDRESS_ID	ADDRESS_NUMBER	ADDRESS_STREET	ADDRESS_TOWN	ADDRESS_STATE	ADDRESS_ZIP
1	1	123	East Main Street	Norton	Massachusetts	12345
2	2	1	West Main Street	Norton	Massachusetts	12345
3	3	2	Leonard Street	Norton	Massachusetts	12345
4	4	3	Plain Street	Norton	Massachusetts	12345
5	5	5	Burt	Norton	Massachusetts	12345
6	6	10	Newland Street	Norton	Massachusetts	12345
7	7	11	Wilbur Street	Norton	Massachusetts	12345
8	8	22	Bay Road	Norton	Massachusetts	12345
9	9	5	Johnson Street	Norton	Massachusetts	12345
10	10	7	Tipping Place	Norton	Massachusetts	12345

I've also created a trigger to ensure the zip code is 5 digits:

```
272 CREATE OR REPLACE TRIGGER address_Zip_trg
273 BEFORE UPDATE OR INSERT ON Address
274 FOR EACH ROW
275 BEGIN
276     IF (LENGTH(:NEW.address_zip) <> 5) THEN
277     RAISE_APPLICATION_ERROR(-20001, 'The Address Zip Code must be 5 digits.');
278     END IF;
279 END;
280 /
```

The next stored procedure adds a person to the database and associates the person to an address. It is named "add_person". This procedure also adds the address to the database if it does not yet exist. I created parameters that correspond to the fields in the person and address tables to allow these to be inputted. The procedure creates two variables, "person_address_id_number" and "person_address_id." The person_address_id_number holds a value for the number of address_id's in the address table that correspond to the address parameter's given. If this number is 0, then an insert statement adds the new address to the address table. If the number is greater than 0, it does nothing. This allows multiple people to be associated to an address without duplicating the address when a new person is added to the database. The next step in the procedure is to add the new person to the person table with the parameters provided. The procedure also uses the parameters of the address in a subquery to associate the person with an address from the address table.

```
186 CREATE OR REPLACE PROCEDURE add_person (
187     P_ssn DECIMAL,
188     p_OLN VARCHAR,
189     p_first_name VARCHAR,
190     p_last_name VARCHAR,
191     p_DOB DATE,
192     P_phone DECIMAL,
193     add_number VARCHAR,
194     add_street VARCHAR,
195     add_town VARCHAR,
196     add_state VARCHAR,
197     add_zip DECIMAL)
198 IS
199     Person_address_id_number DECIMAL(12);
200     Person_address_id DECIMAL(12);
201 BEGIN
202     SELECT count(address_id)
203     INTO Person_address_id_number
204     FROM address
205     WHERE address_number = add_number AND address_street = add_street AND address_zip = add_zip;
206
207     IF Person_address_id_number = 0 THEN
208     INSERT INTO Address (address_id, address_number, address_street, address_town, address_state, address_zip)
209     VALUES (address_seq.nextval, add_number, add_street, add_town, add_state, add_zip);
210     END IF;
211
212     SELECT address_id
213     INTO Person_address_id
214     FROM address
215     WHERE address_number = add_number AND address_street = add_street AND address_zip = add_zip;
216
217     INSERT INTO Person (person_id, address_id, per_SSN, per_oln, per_first_name, per_last_name, per_DOB, per_phone)
218     VALUES (person_seq.nextval, person_address_id, p_ssn, p_oln, p_first_name, p_last_name, p_DOB, p_phone);
219 END;
220 /
```

Script Output

Task completed in 0.088 seconds

```
Procedure ADD_PERSON compiled
```

Here is a screenshot of my stored procedure execution:

```
317 BEGIN
318 ADD_Person (1111111110, 'S12345678', 'John', 'Smith', CAST('1-JAN-1980' AS DATE),
319 5082341234, 123, 'East Main Street', 'Norton', 'Massachusetts', '12345');
320 ADD_Person (1234567890, 'S12225678', 'Helen', 'Smith', CAST('1-MAR-1982' AS DATE),
321 5085557634, 1, 'West Main Street', 'Norton', 'Massachusetts', 12345);
322 ADD_Person (1117771110, null, 'Grace', 'Smith', CAST('11-JAN-2000' AS DATE),
323 5085575434, 2, 'Leonard Street', 'Norton', 'Massachusetts', 12345);
324 ADD_Person (1114444110, 'S16665678', 'Keith', 'Michaels', CAST('1-DEC-2000' AS DATE),
325 5085445234, 3, 'Plain Street', 'Norton', 'Massachusetts', 12345);
326 ADD_Person (1133331110, 'S12344478', 'Robert', 'Jones', CAST('4-NOV-2000' AS DATE),
327 5085976234, 5, 'Burt', 'Norton', 'Massachusetts', 12345);
328 ADD_Person (1122221110, 'S12377778', 'Mary', 'Cota', CAST('17-JUN-2000' AS DATE),
329 5085557774, 10, 'Newland Street', 'Norton', 'Massachusetts', 12345);
330 ADD_Person (1999911110, 'S17775678', 'Jake', 'Boxer', CAST('4-MAY-2000' AS DATE),
331 5085544664, 11, 'Wilbur Street', 'Norton', 'Massachusetts', 12345);
332 ADD_Person (1994444110, 'S15785678', 'Thomas', 'Brigg', CAST('22-AUG-2000' AS DATE),
333 5085235334, 22, 'Bay Road', 'Norton', 'Massachusetts', 12345);
334 ADD_Person (1223911110, 'S12777678', 'Larry', 'Summers', CAST('11-JUL-2000' AS DATE),
335 5085523434, 5, 'Johnson Street', 'Norton', 'Massachusetts', 12345);
336 ADD_Person (1666661110, 'S14776678', 'John', 'Hail', CAST('21-JAN-2000' AS DATE),
337 5081551234, 7, 'Tipping Place', 'Norton', 'Massachusetts', 12345);
338 END;
339 /
340
```

Script Output

Task completed in 0.179 seconds

```
PL/SQL procedure successfully completed.
```

Here is the resulting table:

```
366 SELECT *
367 FROM Person;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.021 seconds

	PERSON_ID	PER_SSN	PER_OLN	ADDRESS_ID	PER_FIRST_NAME	PER_LAST_NAME	PER_DOB	PER_PHONE
1	1	1111111110	S12345678	1	John	Smith	01-JAN-80	5082341234
2	2	1234567890	S12225678	2	Helen	Smith	01-MAR-82	5085557634
3	3	1117771110	(null)	3	Grace	Smith	11-JAN-00	5085575434
4	4	1114444110	S16665678	4	Keith	Michaels	01-DEC-00	5085445234
5	5	1133331110	S12344478	5	Robert	Jones	04-NOV-00	5085976234
6	6	1122221110	S12377778	6	Mary	Cota	17-JUN-00	5085557774
7	7	1999911110	S17775678	7	Jake	Boxer	04-MAY-00	5085544664
8	8	1994444110	S15785678	8	Thomas	Brigg	22-AUG-00	5085235334
9	9	1223911110	S12777678	9	Larry	Summers	11-JUL-00	5085523434
10	10	1666661110	S14776678	10	John	Hail	21-JAN-00	5081551234

I've added 4 triggers for error checking when inputting a person. The first trigger named "person_dog_trg" ensures the person's date of birth is after 1-JAN-1900. This is to ensure that a person is not added to the database with a date of birth that would put them over 120 years old. The second trigger named "person_SSN_trg" ensures the person's social security number is 10 digits. The third trigger named "person_phone_trg" ensures the person's phone number is 10 digits as well. The last

trigger named “check_SSN_redundancy” ensures that no two people are added with the same social security number.

```
242 CREATE OR REPLACE TRIGGER person_DOB_trg
243 BEFORE UPDATE OR INSERT ON person
244 FOR EACH ROW
245 BEGIN
246     IF :NEW.per_DOB < CAST( '1-JAN-1900' AS DATE) THEN
247     RAISE_APPLICATION_ERROR(-20001, 'The date of birth must be after January 1, 1900');
248     END IF;
249 END;
250 /
251
252 CREATE OR REPLACE TRIGGER person_SSN_trg
253 BEFORE UPDATE OR INSERT ON person
254 FOR EACH ROW
255 BEGIN
256     IF (LENGTH(:NEW.per_SSN) <> 10) THEN
257     RAISE_APPLICATION_ERROR(-20001, 'The Social Security number must be 10 digits');
258     END IF;
259 END;
260 /
261
262 CREATE OR REPLACE TRIGGER person_phone_trg
263 BEFORE UPDATE OR INSERT ON person
264 FOR EACH ROW
265 BEGIN
266     IF (LENGTH(:NEW.per_phone) <> 10) THEN
267     RAISE_APPLICATION_ERROR(-20001, 'The phone number must be 10 digits');
268     END IF;
269 END;
270 /
271
```

```
282 CREATE OR REPLACE TRIGGER check_SSN_redundancy
283 BEFORE INSERT OR UPDATE ON Person
284 FOR EACH ROW
285 DECLARE
286     person_ssn_number DECIMAL(1);
287 BEGIN
288     SELECT COUNT(per_ssn)
289     INTO person_SSN_Number
290     FROM Person
291     WHERE :NEW.per_SSN = person.per_ssn;
292
293     IF person_SSN_Number > 0
294     THEN
295        RAISE_APPLICATION_ERROR (-20001, 'This Social Security Number is already in use.');
296     END IF;
297 END;
298 /
```

The third stored procedure I’ve created is built from Use Case 3:

Use Case 3: *Add a Vehicle to the Database Use Case*

4. An officer encounters a vehicle during an incident.
5. The officer enters the vehicle's information into the database.
6. A vehicle is created in the database.

I created a stored procedure named "add_officer". I created parameters that correspond to the fields in the officer table to allow these to be inputted. Inside the stored procedure there is a variable named "off_super_id" which holds the officer_id of the new officer's supervisor. I use this variable in the insert statement to insert the officer_id associated with the supervisor into the new officer record. It also inserts the rest of the parameter data.

```
224 CREATE OR REPLACE PROCEDURE add_officer (
225     O_Badge_id DECIMAL,
226     O_First_Name VARCHAR,
227     O_Last_Name VARCHAR,
228     O_Rank VARCHAR,
229     O_Position VARCHAR,
230     s_badge_id DECIMAL)
231 IS
232     off_super_id DECIMAL(12);
233 BEGIN
234     SELECT officer_id
235     INTO off_super_id
236     FROM officer
237     WHERE s_badge_id = badge_id;
238
239     INSERT INTO Officer (Officer_ID, Supervisor_id, Badge_id, Off_First_Name, Off_Last_Name, Off_Rank, Off_Position)
240     VALUES (officer_seq.nextval, off_super_id, o_Badge_id, o_First_Name, O_Last_Name, O_Rank, O_Position);
241 END;
242 /
```

Script Output × Query Result ×

Task completed in 0.108 seconds

```
Procedure ADD_OFFICER compiled
```

Here is a screenshot of my stored procedure execution, I found that I needed to insert the Chief first, separately from the add_officer procedure as the Chief has no supervisor but is the top of the supervisory pyramid for all other officers:

```
347  INSERT INTO officer
348  VALUES (officer_seq.nextval, null, 134, 'Sean', 'Worrall', 'Chief', 'Administration');
349
350  BEGIN
351  Add_Officer (133, 'Michael', 'Jacobs', 'Lieutenant', 'Patrol', 134);
352  Add_Officer (123, 'Jake', 'Johnson', 'Sergeant', 'Patrol', 133);
353  Add_Officer (333, 'Patrick', 'Booher', 'Patrolman', 'Patrol', 123);
354  Add_Officer (130, 'Robert', 'Goodwin', 'Patrolman', 'Patrol', 123);
355  Add_Officer (143, 'Ashley', 'Kennedy', 'Patrolman', 'Patrol', 123);
356  Add_Officer (190, 'Brian', 'Clark', 'Sergeant', 'Dectective', 134);
357  Add_Officer (173, 'Nancy', 'Leeland', 'Patrolman', 'Dectective', 190);
358  Add_Officer (564, 'James', 'Murphy', 'Sergeant', 'Patrol', 134);
359  Add_Officer (190, 'Janna', 'Perez', 'Patolman', 'Patrol', 564);
360  END;
361  /
```

Script Output | Query Result

Task completed in 0.159 seconds

```
Procedure ADD_OFFICER compiled


1 row inserted.


PL/SQL procedure successfully completed.
```

The resulting table:

```
369  SELECT *
370  FROM officer;
371
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.011 seconds

	OFFICER_ID	SUPERVISOR_ID	BADGE_ID	OFF_FIRST_NAME	OFF_LAST_NAME	OFF_RANK	OFF_POSITION
1	1	(null)	134	Sean	Worrall	Chief	Administration
2	2	1	133	Michael	Jacobs	Lieutenant	Patrol
3	3	2	123	Jake	Johnson	Sergeant	Patrol
4	4	3	333	Patrick	Booher	Patrolman	Patrol
5	5	3	130	Robert	Goodwin	Patrolman	Patrol
6	6	3	143	Ashley	Kennedy	Patrolman	Patrol
7	7	1	190	Brian	Clark	Sergeant	Dectective
8	8	7	173	Nancy	Leeland	Patrolman	Dectective
9	9	1	564	James	Murphy	Sergeant	Patrol
10	10	9	190	Janna	Perez	Patolman	Patrol

Query Executions and Explanations

The first question I identified is, "What are the names and addresses of the victim of Domestic Disturbance incidents over the last year and what were their relationships to the offender?"

This question is an important question to answer for a police department because the role of the department is not only to respond to incidents but to attempt to prevent them. A major aspect of modern policing is community outreach. Many departments are adding clinicians to help victims and prevent further incidents from developing. The clinicians follow up with incidents such as domestic violence or mental health crises. It is important for the department to identify who the victims are. The clinicians then can follow up with them to advise them of their options and help them seek treatment.

The second question I identified is "In the past year, who were the defendants in criminal cases that were a result of motor vehicle stops? What were the citation numbers and the case status?"

This is an important question for incident history and criminal case tracking. The department assigns an officer as the court officer who must track all criminal cases, coordinate with the court to have officer testify in court and represent officers when they cannot attend. The court officer must track criminal citations as they go through the registry of motor vehicles and the court system. The court officer must follow up on each case, update the status to track the case progress, and schedule officers to go to court when needed.

The third question I identified is, "How many arrest occurred in 2020 that were a result of domestic violence incidents?"

This question is important because departments must keep track of all statistics required by law and make them available to the public. It is also important to forecast officer staffing requirements, and resource planning. Additionally, to tie back into the earlier question of who the victims of domestic violence were, it is critical to track trends in crime and provide help to victims and offenders. If there is a significant increase in arrests for domestic violence it may require the departments administration to change tactics and allocate more of the budget to community outreach events on educating the public or adding more clinicians to help offenders and victims seek treatment.

Query Executions and Explanations

"What are the names and addresses of the victim of Domestic Disturbance incidents over the last year and what were their relationships to the offender?"

```
--What are the names and addresses of the victims of Domestic Disturance incidents
--over the last year and what were their relationships to the offender?
SELECT Person.per_first_name, person.per_last_name, victim_to_offender
AS Victim_Relationship_to_Offender, address.address_number,
address.address_street, address.address_town, address.address_state, address.address_zip
FROM Person
JOIN Address ON person.address_id = address.address_id
JOIN Participant ON participant.person_id = person.person_id
JOIN victim ON participant.participant_id = victim.participant_id
JOIN Incident ON participant.incident_id = incident.incident_id
WHERE Inc_Category = 'Domestic Disturbance' AND Is_Victim_flag = 'Y';
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 3 in 0.024 seconds

	PER_FIRST_NAME	PER_LAST_NAME	VICTIM_RELATIONSHIP_TO_OFFENDER	ADDRESS_NUMBER	ADDRESS_STREET	ADDRESS_TOWN	ADDRESS_STATE	ADDRESS_ZIP
1	Helen	Smith	Wife	1	West Main Street	Norton	Massachusetts	12345
2	Robert	Jones	Cousin	5	Burt	Norton	Massachusetts	12345
3	Thomas	Brigg	Father	22	Bay Road	Norton	Massachusetts	12345

This query selects the person's first name, last name, relationship to the offender, and their address. It joins five tables together (Person, Address, Incident, Victim, and Participant) and uses the WHERE clause to limit the results to two conditions: Where the incident category is "Domestic Disturbance" and the Victim flag in incident is "Y". The query pulls the person's name from the Person table, the victim flag

from the participant table, the address from the address table, the incident category from the incident table, and the relationship to the offender from the victim table.

"In the past year, who were the defendants in criminal cases that were a result of motor vehicle stops? What were the citation numbers and the case status?"

```
--This Query answers this question:
--In the past year, who were the defendants in criminal cases that were a result of a motor vehicle stop, what were the citation numbers and the case status?
SELECT Person.per_first_name, person.per_last_name, person.per_oln, cite_num, cite_status
FROM PERSON
JOIN Citation ON citation.person_id = person.person_id
JOIN Incident ON incident.incident_id = citation.incident_id
JOIN Cite_criminal ON citation.cite_id = cite_criminal.cite_id
WHERE Inc_Category = 'M/V Stop' AND Inc_Date_Time > CAST('31-DEC-2019' AS DATE) AND Inc_Date_Time < CAST('1-JAN-2021' AS DATE);
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 2 in 0.028 seconds

	PER_FIRST_NAME	PER_LAST_NAME	PER_OLN	CITE_NUM	CITE_STATUS
1	Keith	Michaels	S16665678	B87754	Open
2	Robert	Jones	S12344478	B77754	Open

This query selects the person's full name, their OLN (operator/driver's license number), the citation number and the case status. It joins four tables: Person, Citation, Cite_Criminal, and Incident. It limits the results with the WHERE clause using the condition that the incident category is "M/V Stop" (a motor vehicle stop) and uses a date range to ensure the results are from 2020. It obtains the person's name and OLN from the person table, the citation number from the citation table, and the case status from the Cite_criminal table.

"How many arrest occurred in 2020 that were a result of domestic violence incidents?"

```
--This Query answers the question:
--How many arrests were made in 2020 that occured as a result of Domestic Disturbance incidents?
SELECT COUNT(Offender_Arrested) AS Domestic_Disturbance_Arrests_In_2020
FROM Offender
JOIN Participant ON participant.participant_id = offender.participant_id
Join Incident ON Incident.Incident_id = participant.incident_id
WHERE Inc_Category = 'Domestic Disturbance' AND offender_arrest_date > CAST('31-DEC-2019' AS DATE)
AND offender_arrest_date < CAST('1-JAN-2021' AS DATE) AND  Offender_arrested = 'Y';
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 1 in 0.002 seconds

	DOMESTIC_DISTURBANCE_ARRESTS_IN_2020
1	2

This query selects the number of arrests resulting from Domestic Disturbance calls. The query joins three tables: Offender, Participant, and Incident. It limits the results with a WHERE clause and the condition that the incident category is "Domestic Disturbance," that the offender_ arrested is "Y", and has a date range to ensure the incident occurred in 2020.

Index Identification and Creations

Primary Keys that are already indexed:

Cite_fee.citation_id
Cite_warning_citation_id
Cite_criminal. citation_id
Citation.citation_id
Caller.participant_id
Victim. participant_id
Offender.participant_id
Witness.participant_id
Participant.participant_id
Responder.responder_id
Involved_vic.involved_vic_id
Vehicle.vehicle_id
Incident.incident_id
Person.person_id
Address.address_id
Officer.officer_id

Foreign Keys to be indexed:

Column	Unique?	Description
Officer.Off_Supervisor (Already indexed with Officer.officer_id)	Not Unique	Foreign key (recursive) references officer (officer_id). This is the officer's direct supervisor. It's not unique because an officer can be a supervisory to more than one other officer. This column is recursive to Officer, and utilizes the Officer_id which is already indexed.
Person.address_id	Not Unique	The foreign key in Person referencing Address is not unique because there can be many people with the same address.
Incident.address_id	Not Unique	The foreign key in Incident referencing Address is not unique because there can be many Incidents with the same address.
Incident.officer_id	Not Unique	The foreign key in Incident referencing Officer is not unique because there can be many Incidents with the same primary officer.
Involved_vic.incident_id	Not Unique	The foreign key in Involved_Vic referencing Incident is not unique because a vehicle can be involved in many different incidents.
Involved_vic.vehicle_id	Not Unique	The foreign key in Involved_Vic referencing Vehicle is not unique because a vehicle can be involved in many different incidents.
Responder.incident_id	Not Unique	The foreign key in Responder referencing Incident is not unique because an incident can own many different responders.

Responder.officer_id	Not Unique	The foreign key in Responder referencing Officer is not unique because an officer can be many different responders.
Participant.person_id	Not Unique	The foreign key in Participant referencing person is not unique because a person can own many different participants.
Participant.incident_id	Not Unique	The foreign key in Participant referencing incident is not unique because an incident can own many different participants.
Citation.officer_id	Not Unique	The foreign key in Citation referencing Officer is not unique because each Officer can write many different citations.
Citation.vehicle_id	Not Unique	The foreign key in Citation referencing Vehicle is not unique because a Vehicle can be associated with many citations.
Citation.person_id	Not Unique	The foreign key in Citation referencing Person is not unique because a Person can be associated with many citations.
Citation.incident_id	Not Unique	The foreign key in Citation referencing Incident is not unique because an Incident can be associated with many citations.

From the queries I created, I identified these fields from the WHERE clauses which I believe will be important for limiting data and used often by members of the police department. The incident dates and the incident categories will both be very important for many different queries. The arrest dates will also be an important column for generating queries to track arrests. The citation type can be utilized to differentiate between citations in queries. The other columns utilized are primary keys and are already indexed.

Incident.inc_date_time	Not Unique	This field is important for limiting incidents by date and is not unique because there may be multiple incidents with the same date.
Incident.inc_category	Not Unique	This field is important for limiting incidents by category is not unique because there may be multiple incidents with the same date.
Citation.Cite_type	Not Unique	This field is important for limiting citation down by type.
Offender.Offender_arrest_date	Not Unique	This field is important for limiting arrest column by dates. It is not unique as there may be many arrests on the same date.

```
--INDEXES
CREATE INDEX Person_address_id_idx
ON Person(address_id);
CREATE INDEX incident_address_id_idx
ON incident(address_id);
CREATE INDEX incident_officer_id_idx
ON incident(officer_id);
CREATE INDEX involved_vic_incident_id_idx
ON involved_vic(incident_id);
CREATE INDEX involved_vic_vehicle_id_idx
ON involved_vic(vehicle_id);
CREATE INDEX responder_incident_id_idx
ON responder(incident_id);
CREATE INDEX responder_officer_id_idx
ON responder(officer_id);
CREATE INDEX participant_person_id_idx
ON participant(person_id);
CREATE INDEX participant_incident_id_idx
ON participant(incident_id);
CREATE INDEX citation_officer_id_idx
ON citation(officer_id);
CREATE INDEX citation_vehicle_id_idx
ON citation(vehicle_id);
CREATE INDEX citation_incident_id_idx
ON citation(incident_id);
CREATE INDEX citation_person_id_idx
ON citation(person_id);
CREATE INDEX incident_inc_date_time_idx
ON Incident(inc_date_time);
CREATE INDEX incident_inc_category_idx
ON Incident(inc_category);
CREATE INDEX citation_cite_type_idx
ON citation(cite_type);
CREATE INDEX offender_arrest_date_idx
ON offender(offender_arrest_date);
```

Script Output

Task completed in 0.193 seconds

```
Index CITATION_CITE_TYPE_IDX created.


Index OFFENDER_ARREST_DATE_IDX created.
```

History Table Demonstration

Explain the specifics of your history table, including how the trigger works, and demonstrate that the history table captures changes.

In reviewing my physical ERD, I determined that the citation_criminal status attribute would benefit from a history table to track the status changes as the criminal case went through the court system. The officer in charge of tracking court cases would be able to track the status changes over time and the dates of the changes. This history would also help determine the progress of a case, any necessary paperwork or evidence that would need to be submitted depending on the status of the case or dates requiring officers to be summoned into court to testify.

My new structural rule is: Each cite_criminal may have one or more status_changes; each status change is associated to one cite_criminal.

The relationship from the perspective of cite_criminal is plural and optional since a cite_criminal may have none or many status_changes. From the perspective of status_change the relationship is singular and mandatory since each status_change must associate to only one cite_criminal.

The Status_Change entity was added to the conceptual and physical ERD. Below are the attributes I added and why.

Attribute	**Description**
Status_Change_Id	This is the primary key for the history table. It is a DECIMAL(12) to allow for many values
Old_Status	This is the old status of the cite_criminal before the change. The datatype mirrors the status datatype in the cite_criminal table.
New_Status	This is the new status of the cite_criminal before the change. The datatype mirrors the status datatype in the cite_criminal table.
Cite_ID	This is a foreign key to the cite_criminal table, a reference to the criminal citation that had the change in status.
Change_Date	This is the date the citation status change occurred with a DATE datatype.

Here is a screenshot of my table and sequence creation, which has all the same attributes and datatypes as indicated in the DBMS physical ERD:

```
161 CREATE TABLE Status_Change (
162  Status_Change_ID DECIMAL(12) PRIMARY KEY,
163  Cite_ID DECIMAL(12),
164  Old_Status VARCHAR(25),
165  New_Status VARCHAR(25),
166  Change_Date DATE,
167  CONSTRAINT Status_Change_cite_id_FK FOREIGN KEY (Cite_id) REFERENCES Citation(cite_id));
```

```
179  CREATE SEQUENCE Status_Change_seq START WITH 1;
```

Here is a screenshot of my trigger creation which will maintain the Status_Change table:

```
346 CREATE OR REPLACE TRIGGER Status_Change_trg
347 BEFORE UPDATE OF Cite_Status ON Cite_Criminal
348 FOR EACH ROW
349 BEGIN
350     INSERT INTO Status_Change (Status_Change_ID, Cite_id, Old_Status, New_Status, Change_Date)
351     VALUES (Status_Change_seq.NEXTVAL, :NEW.Cite_id, :OLD.Cite_Status, :New.Cite_Status, trunc(sysdate));
352 END;
353 /
```

Script Output | Task completed in 0.054 seconds

```
Trigger STATUS_CHANGE_TRG compiled
```

To test out the trigger, I've created two criminal citations in the cite_criminal table:

```
682 SELECT *
683 FROM Cite_Criminal;
```

Script Output | Query Resu | SQL | All Rows Fe

	CITE_ID	CITE_STATUS
1	2	Open
2	5	Open

I changed the status 3 times one of the citations to discovery, pre-trial, and trial as demonstrated below.

```
685 Update Cite_Criminal
686 SET cite_status = 'Discovery'
687 WHERE Cite_id = 2;
688
689 Update Cite_Criminal
690 SET cite_status = 'Pre-Trial'
691 WHERE Cite_id = 2;
692
693 Update Cite_Criminal
694 SET cite_status = 'Trial'
695 WHERE Cite_id = 2;
696
```

Script Output | Task completed in 0.098 sec

```
1 row updated.


1 row updated.


1 row updated.
```

Lastly, I verified that my trigger worked as expected by selecting all from the Status_Change table:

```
697  SELECT *
698  FROM Status_Change;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.029 seconds

	STATUS_CHANGE_ID	CITE_ID	OLD_STATUS	NEW_STATUS	CHANGE_DATE
1	1	2	Open	Discovery	24-APR-22
2	2	2	Discovery	Pre-Trial	24-APR-22
3	3	2	Pre-Trial	Trial	24-APR-22

The results demonstrate that the status went from open to discovery, to pre-trial, then to trial, all for the citation with cite_id 2.

Data Visualizations

An important piece of data for a police department is to track the number of arrests over time, but to also analyze that information to look for trends. There may be different times of the year that have an uptick in arrests, or even times of a month where the number of arrests is higher. A department would be able to utilize this information to plan either having more officers on duty, more community outreach as awareness or to be more proactive during these times. I asked the question, "How many arrests were made per day in the month of July 2020?"

To answer this question, I developed and executed a query that counts the number of arrests per day in July 2020. The results are ordered by date. The query and results are shown below.

```
725  --How many arrests by date were made in July of 2020?
726  SELECT Inc_Date_Time AS Arrest_Date, COUNT(Offender_Arrested) AS Number_of_Arrests
727  FROM Offender
728  JOIN Participant ON participant.participant_id = offender.participant_id
729  Join Incident ON Incident.Incident_id = participant.incident_id
730  WHERE offender_arrest_date > CAST('30-JUN-2020' AS DATE)
731  AND offender_arrest_date < CAST('1-AUG-2020' AS DATE) AND Offender_arrested = 'Y'
732  GROUP BY Inc_Date_time, Offender_Arrested
733  ORDER BY Inc_Date_time;
734
```

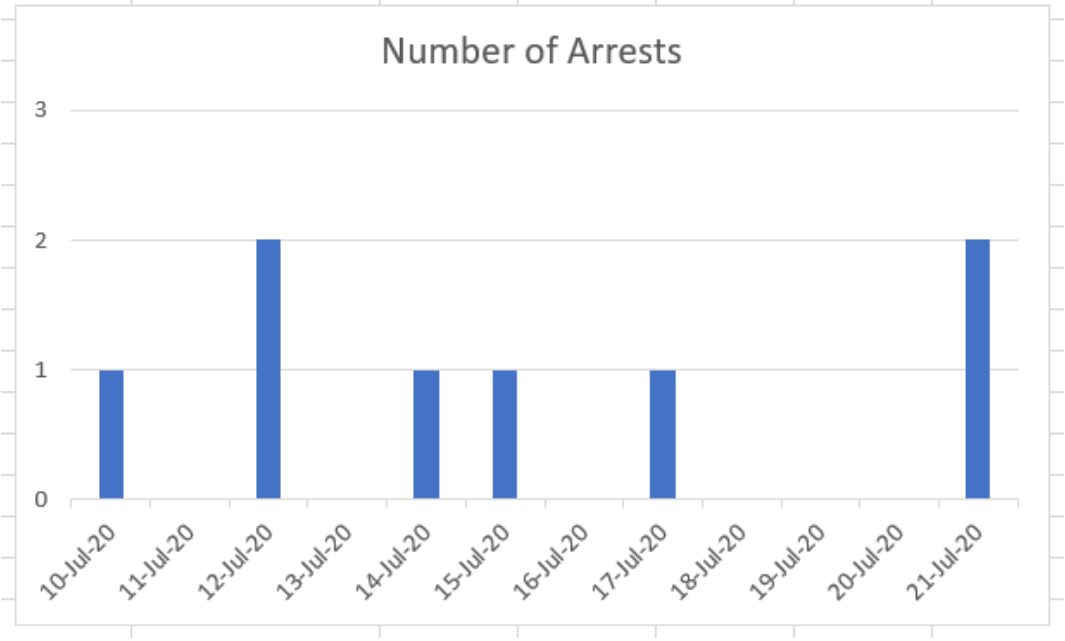
Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 6 in 0.094 seconds

	ARREST_DATE	NUMBER_OF_ARRESTS
1	10-JUL-20	1
2	12-JUL-20	2
3	14-JUL-20	1
4	15-JUL-20	1
5	17-JUL-20	1
6	21-JUL-20	2

Since these results are only utilizing one measure, I decided to use a bar chart to visualize the results. After exporting to a CSV file and creating the bar chart in excel, these are the following results:

Date	Number of Arrests
10-Jul-20	1
12-Jul-20	2
14-Jul-20	1
15-Jul-20	1
17-Jul-20	1
21-Jul-20	2



There are multiple pieces of information that can be obtained from this visualization. The first thing observed is that there were no arrests made in July 2020 prior to July 10, 2020. Additionally, no arrests were made in July after July 21, 2020. All the arrests were made between July 10 and July 21, 2020. It also shows that no day had more than 2 arrests made for the month. It would be interesting to compare this bar chart to July of 2019 or July of 2018 to observe any trends year over year.

From this information, a department would be able to utilize it in a few different ways. Since there is not a single day that jumps out with the number of arrests, it appears that the department would not have to worry about over staffing on a particular day in July to handle the volume of arrests. It also shows that the volume or arrests over the month is not overwhelming, which means that the department may be able to pull officers from regular patrol in July and assign them to other tasks such as community engagement or traffic enforcement without having to worry about understaffing a shift to handle arrest bookings.

A second important visualization that would be useful to a police department, and especially the court officer, is how many criminal citations changed to a status of Trial by day in a certain month. I chose to ask the question, “how many criminal citations changed to a status of Trial by day in the month of July 2020?” This question can give the court officer an idea of how many trials he would have to schedule officers to be at.

To answer this question, I developed and executed a query that counts the number of criminal citations that changed status to Trial per day in July 2020. The results are ordered by date. The query and results are shown below.

```
809  --This Query answers the question:
810  --How many Criminal Citations changed status to Trial in July 2020 by date?
811  SELECT change_date, COUNT(New_Status)
812  FROM Status_Change
813  WHERE CHANGE_Date > CAST('30-JUN-2020' AS DATE)
814  AND change_date < CAST('1-AUG-2020' AS DATE) AND New_Status = 'Trial'
815  GROUP BY change_date
816  ORDER BY change_date;
817
```

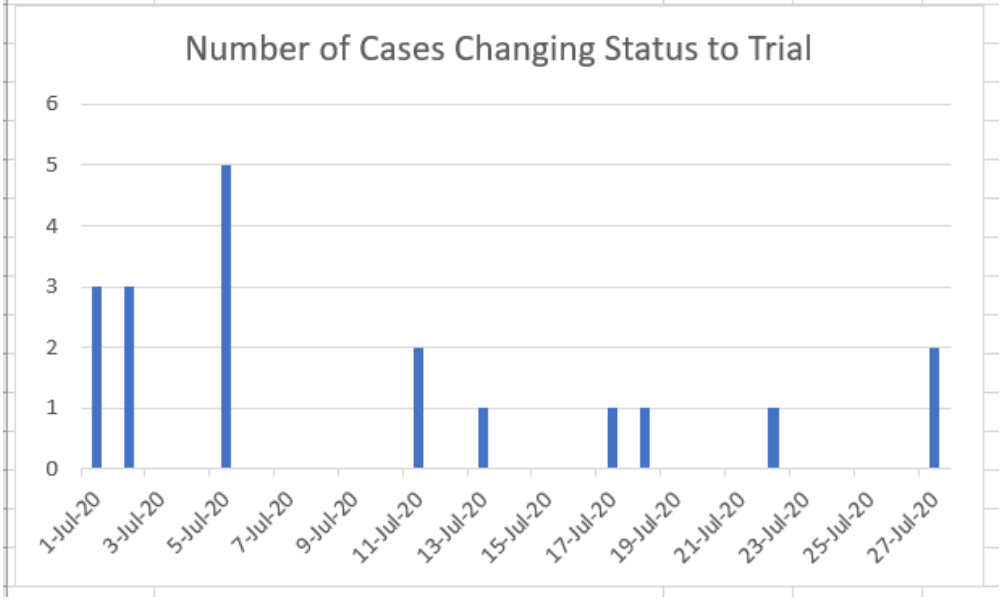
Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 9 in 0.017 seconds

	CHANGE_DATE	COUNT(NEW_STATUS)
1	01-JUL-20	3
2	02-JUL-20	3
3	05-JUL-20	5
4	11-JUL-20	2
5	13-JUL-20	1
6	17-JUL-20	1
7	18-JUL-20	1
8	22-JUL-20	1
9	27-JUL-20	2

Since these results are only utilizing one measure, I decided to use a bar chart to visualize the results. After exporting to a CSV file and creating the bar chart in excel, these are the following results:

Date	Number of Cases Changing Status to Trial
1-Jul-20	3
2-Jul-20	3
5-Jul-20	5
11-Jul-20	2
13-Jul-20	1
17-Jul-20	1
18-Jul-20	1
22-Jul-20	1
27-Jul-20	2



These results show that several cases changed status during the month of July 2020. The results also show that the number of cases changing status is not concentrated in any specific timeframe within the month, but in the beginning of the month there was a larger number of cases changing status per day than there was in the middle or end of the month. This could indicate that the court is busier in the beginning of the month with trials than the middle or end. The court officer could use this information to notify officers that during the beginning of the months, they are more likely to have court dates and to plan their schedules accordingly.

Summary and Reflection

Iteration 1:

My database is going to be designed to support a police department's records management system called Police Source. It will provide the department with the ability to track incidents that occur in its jurisdiction and keep records of the vast amount of data associated. These records can be utilized for a multitude of different purposes from officer's using it while dealing with members of the public or during an incident. The command staff can utilize the information to make informed decisions on staffing and high crime areas, or to satisfy information requests from the public. The database must be able to support officers entering information regarding people, vehicles, incidents, and citations, and be

able to search for specific records when needed. It must also provide the command staff an ability to analyze the data.

I am excited to build this database because it will provide me a more comprehensive understanding of on an application that I use daily and one I understand from a user's perspective. It will be extremely interesting to work to put it together from the database perspective and learn how this data is actually stored to facilitate its efficient and timely use. I think this will be a very interesting topic that relates directly to my current career.

A few concerns that I have are with the complexity that can inherently result from this type of data. As I think about the number of elements that go into each area of the use cases I described above, there is a lot of interrelated aspects between the use cases and between incidents, vehicles, people, officers, and citations. I am nervous that this will be very difficult to model and require a number of tables for associations, which I know some information about but do not fully understand yet. I tried to limit the use cases to officers, people, incidents, and citations as I believe further functionality that the application could handle such as bookings, arrests, restraining orders, domestic violence follow ups, and other such features would be too complicated for the scope of this class. Ultimately, I believe that I will overcome my concerns as I learn more about databases and get into the later weeks of this course.

Iteration 2:

The structural database rules and the ERD that I've created contain eight entities that I identified through my five uses cases: Officer, Person, Vehicle, Involved_vic, Incident, Participant, Responder, and Citation. These eight entities led to thirteen different structural rules. This really surprised me. I had originally thought, during iteration 1, that I would only end up with five entities that were easily apparent when writing the use cases. While analyzing my use cases to identify structural rules, I discovered a few of the entities had many-to-many relationships. To accurately display them in the ERD, I created additional entities as composite entities with one-to-many relationships. These entities also required structural rules leading to the final number of thirteen. This exercise really showed me how complicated a database can become. I was only working with five use cases, but I can imagine how many entities would be required if the number of use cases started to grow.

I also found the structural rules I defined to be essential and very straightforward when designing the ERD. I went iteratively though the structural rules to help my build the ERD, then fine-tuned it to be visually acceptable and all relationship legible. I do feel it will become more cluttered and possibly more complicated as I add attributes and primary and foreign key, but I am excited to continue to build the database going forward.

Iteration 3:

I felt that determining specialization-generalization relationships in this iteration was a good exercise in analyzing what functionality the database required and specifically what would add value. I found that there were a few use cases and entities I considered immediately to develop supertype and subtype relationships for, but ultimately decided against it. As I looked through my use cases and the entities developed in iteration 2, I identified Citation and Participation as candidates for supertypes, but I did not believe that the other entities identified would benefit from specialization-generalization relationships. I felt these additional relationships would not add additional value to the database.

I considered adding supertype and subtype relationships to the Incident entity, but as I thought more about what value the relationship and additional tables would add, I determined that the subtypes of incident would not require additional attributes beyond the attributes the Incident entity table would already have. I believe that an application level drop-down list of incident type would better limit redundancy in the database than creating incident subtype tables. I feel that these additional tables would only add data redundancy.

I also considered adding Vehicle or Involved_Vic subtypes, but I again did not think this relationship would add value to the database. I did not see the type of vehicle affecting behavior or adding more functionality beyond a field in the Vehicle table. In terms of the Police Source application, the type of vehicle, being a motorcycle, tractor trailer, or passenger car, would not be an important aspect beyond another application level drop-down selection of vehicle type. A citation would not change due to the type of vehicle, and I do not believe there would be additional attributes required for subtypes beyond the attributes already present in Vehicle or Involved_Vic supertypes.

Once I started to build the initial physical ERD, I felt I understood what relationships I wanted to visualize but I did find it challenging to show the relationships without overlapping relationship lines or having lines cross through other entities. Regarding the mapping of foreign keys, the work I did in Iteration 2 to define the relationship plurality eased the burden of determining which table the foreign keys would reside in and it became more of a mechanical exercise. I also felt that building bridge tables in the previous iteration helped simplify the process as it was a step I had already completed. I look forward to the next iteration of fully defining the attributes and starting to create the tables in SQL.

Iteration 4:

When determining my attributes for each entity, I decided to make a change to the Participant specialization-generalization relationship. I determined that the involved_party subclass of Participant was redundant as it did not require any additional fields beyond the superclass', Participant, attributes. A participant of an incident is an involved party by nature as it is a person associated with an incident. If a person is associated as a Participant in an incident, they are already an involved party, and do not need to be put into a subclass. I decided to remove that subclass, but to allow a person to be associated to an incident as only a Participant and not necessarily a caller, witness, victim, or offender, I changed the Participant superclass relationship to the subclasses to partial and optional on the EERD.

When determining attributes for Citation, I did not identify any specific attributes for a written warning citation, but I left it as a subclass in the Physical ERD as I believe there could be attributes that may be identified in the future.

I've had some difficulty with the concepts of Normalization, but I believe that I have been able to work through them with the aid of the textbook, online lectures, and the live classrooms. I feel that I've normalized the tables to BCNF in all besides the address table. I did notice that in my vehicle table, I could normalize it to 4NF by removing color from the table as it would be an independent list. I wasn't sure if removing this type of redundancy would outweigh the added complexity to queries.

During the normalization process, I created a few new entities to include an address table, Social_Security, and OLN. The address table is referenced by both Incident and Person entities. I've gone back and created structural rules for this entity and added it to the conceptual and physical EERD. I did not bring the address table to BCNF as I did not feel that having the number of additional tables to achieve full BCNF would add any benefit to the database against the level of complexity it would create.

I removed the Social_Security and OLN attributes from the Person entity as they both could be determinants, but neither work as candidate keys as they are both nullable. A person may not have a

social security number or an operator's license. I've created references to these as foreign keys in the person table. I've also updated my use cases and structural rules to reflect these new entities as well as my conceptual ERD.

I found that my Physical EERD has grown much larger and more complex. I exported a PDF from Lucid Chart and imbedded it into this document, but it appears to be very small, so I uploaded a PDF of it as well. Due to the number of foreign keys and associations between entities, I decided to user ALTER TABLE commands in my SQL commands to avoid having to determine the table creation order. I was a bit confused on creating the subclasses to Citation and Participant super classes, but I made the superclass primary key the primary key of each subclass table and created a foreign key back to the Superclass.

I found that this iteration was quite challenging in both the new concept of normalization and determining what fields would be important for each entity. I believe that the Police Source database does comprise the important fields that would be utilized daily or multiple times a day, and limits redundancy as much as possible without creating too much complexity. I look forward to your feedback and adding data and defining queries to Police Source next iteration.

Iteration 5:

After reviewing the Iteration 4 feedback, I've decided it does make sense to denormalize the Social_Security and OLN entities and instead make them attributes in the Person table. I agree that this would decrease the complexity and better achieve the normalization goal. I wnet back through and updated the document with these changes. I also went back through my SQL table creation and changed it to incorporate the foreign keys in the table creation.

I created three stored procedures for this iteration which I found very useful when inputting data into my database. I did struggle for quite a while on the add_person stored procedure as I worked through adding a person and associating them with an address. I knew that I did not want to duplicate any address, but I also wanted a way to add a person whose address was not yet in the database. I feel that my solution works well for this, but it took a lot of trial and error to create it. I also created an add_address and add_officer stored procedures. The add_officer, since it uses a variable to subquery for the supervisors' officer_id, required me to input the chief first to be able to build the hierarchy. I wasn't sure how to build this procedure if there was a null for the supervisor, but besides the chief, all others would have a supervisor.

As for inputting the actual data in the script, there was quite a few foreign keys that I had to make sure that I was referencing back to the correct one in the reference table. For example, with a citation, I had to make sure the incident it referenced was a motor vehicle stop and that there was a participant as the offender. Once this was done, I was able to formulate my questions and create the SQL to accomplish them.

I felt that the index process was straight forward once I determined the foreign keys and the columns that corresponded with the where clauses in my queries. I believe that as I continue to use the database, I will determine more columns that would benefit from indexing. It was very excited to be able to see data in the tables that I created and being able to apply queries to answer questions that a police organization would commonly ask.

Iteration 5:

After reviewing the feedback from iteration 5, I adjusted my third query to reflect the changes requested by putting the offender_arrested = 'y' into the WHERE clause and removed the GROUP BY and HAVING.

This iteration, I updated my conceptual and physical ERD to reflect the addition of the Status_Change history table. I also updated the structural rules to reflect the new entity and its relationship to cite_criminal.