FALL 2016

Far Hills – Bedminster Fire Department

Volunteers Serving the Borough of Far Hills and the Township of Bedminster Since 1901

29-FIRE

DATABASE PROPOSAL

ROBERT LYNCH
8 DECEMBER 2016
EMERGENCY SOLUTIONS
3399 North Road Poughkeepsie, NY 12601

Table of Context

Executive Summary

Overview

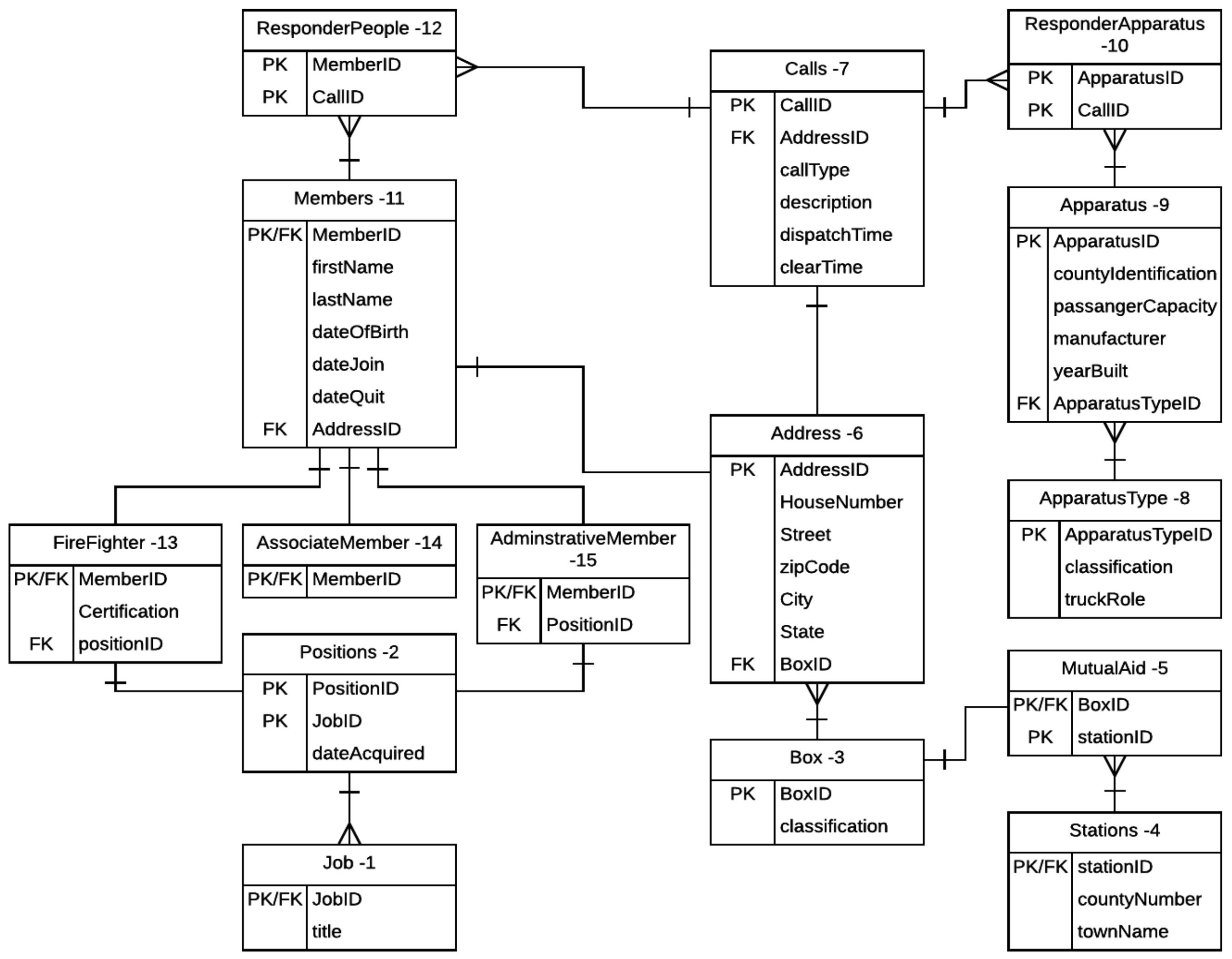
Located in the town of Bedminster, New Jersey, the Far Hills-Bedminster Fire Department responds to calls in Bedminster, Far Hills, and Pluckemin. The fire department has about 500 calls per year, making it hard to keep track on paper. The department wants to be able to keep track of their members, calls, and apparatus in one all-encompassing database. This database will be used to keep statistics such as member response, call volume, and man-hours.

Far Hills-Bedminster Fire Department is a non-profit agency. All information in this database is depicted and is fictitious. Any similarities are mealy a coincidence.

Objectives

The purpose of this document is to give an overview of the database. This document explains how the database is designed and implemented. Below is an entity relation diagram as well as SQL statements that implement, add data, create views, generate reports, create store procedures, create triggers, and add security to the database. This database was designed in PostgresSQL 9.5.

Entity Relation Diagram



Tables

Calls

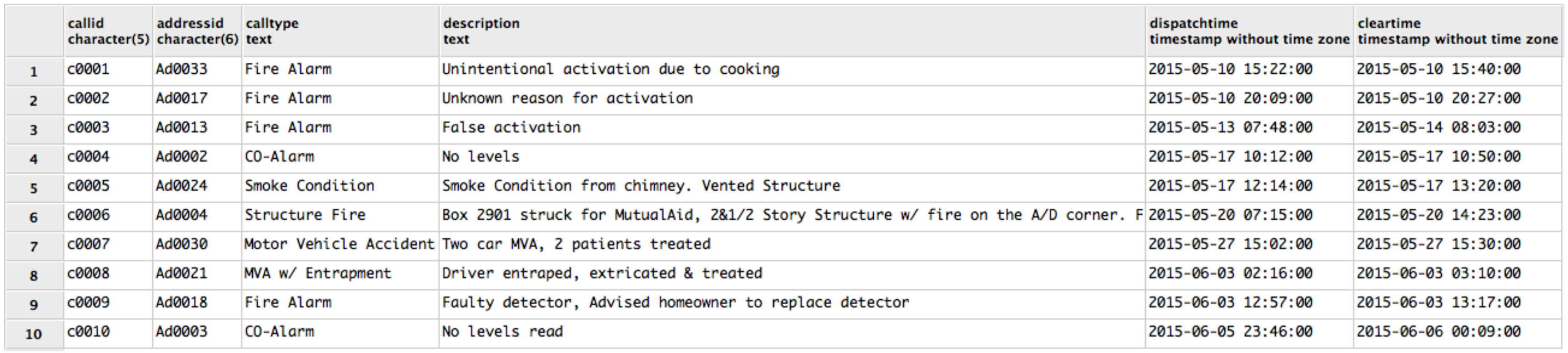
This table contains information about a fire call. It is linked the table ResponderPeople, ResponderApparatus, and Address so that the database can keep track members who responded, apparatus that responded, and a detailed (and atomic) address of the call.

SQL Code

```sql
CREATE TABLE calls(
    callID char(5) NOT NULL unique,
    addressID char(6) references address(addressID),
    callType text,
    description text,
    dispatchTime timestamp NOT NULL,
    clearTime timestamp check(clearTime>dispatchTime) NOT NULL,
    primary key (callID)
);
```

Functional Dependency

callID → AddressID, callType, description, dispatchTime, clearTime



	callid character(5)	addressid character(6)	calltype text	description text	dispatchtime timestamp without time zone	cleartime timestamp without time zone
1	c0001	Ad0033	Fire Alarm	Unintentional activation due to cooking	2015-05-10 15:22:00	2015-05-10 15:40:00
2	c0002	Ad0017	Fire Alarm	Unknown reason for activation	2015-05-10 20:09:00	2015-05-10 20:27:00
3	c0003	Ad0013	Fire Alarm	False activation	2015-05-13 07:48:00	2015-05-14 08:03:00
4	c0004	Ad0002	CO-Alarm	No levels	2015-05-17 10:12:00	2015-05-17 10:50:00
5	c0005	Ad0024	Smoke Condition	Smoke Condition from chimney. Vented Structure	2015-05-17 12:14:00	2015-05-17 13:20:00
6	c0006	Ad0004	Structure Fire	Box 2901 struck for MutualAid, 2&1/2 Story Structure w/ fire on the A/D corner. F	2015-05-20 07:15:00	2015-05-20 14:23:00
7	c0007	Ad0030	Motor Vehicle Accident	Two car MVA, 2 patients treated	2015-05-27 15:02:00	2015-05-27 15:30:00
8	c0008	Ad0021	MVA w/ Entrapment	Driver entraped, extricated & treated	2015-06-03 02:16:00	2015-06-03 03:10:00
9	c0009	Ad0018	Fire Alarm	Faulty detector, Advised homeowner to replace detector	2015-06-03 12:57:00	2015-06-03 13:17:00
10	c0010	Ad0003	CO-Alarm	No levels read	2015-06-05 23:46:00	2015-06-06 00:09:00

Sample output from Calls

Address

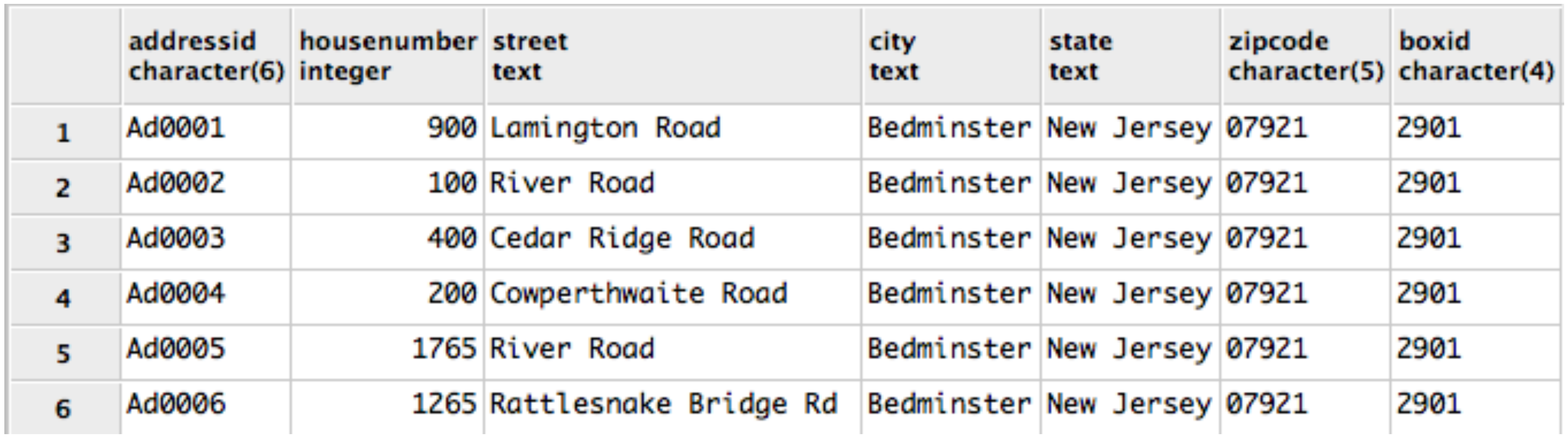
This table contains information on addresses in the database. It is shared with calls and members. Each address has a boxID. Boxes are explained later in this document. All the columns in this table are atomic.

SQL Code

```
CREATE TABLE address(
    addressID char(6) NOT NULL unique,
    houseNumber int,
    street text,
    zipCode char(5),
    city text,
    state text,
    boxID char(4) references box(boxID),
    primary key (addressID)
);
```

Functional Dependency

addressID → houseNumber, street, city, state, zipCode, boxID



	addressid character(6)	housenumber integer	street text	city text	state text	zipcode character(5)	boxid character(4)
1	Ad0001	900	Lamington Road	Bedminster	New Jersey	07921	2901
2	Ad0002	100	River Road	Bedminster	New Jersey	07921	2901
3	Ad0003	400	Cedar Ridge Road	Bedminster	New Jersey	07921	2901
4	Ad0004	200	Cowperthwaite Road	Bedminster	New Jersey	07921	2901
5	Ad0005	1765	River Road	Bedminster	New Jersey	07921	2901
6	Ad0006	1265	Rattlesnake Bridge Rd	Bedminster	New Jersey	07921	2901

Sample output from Address

Box

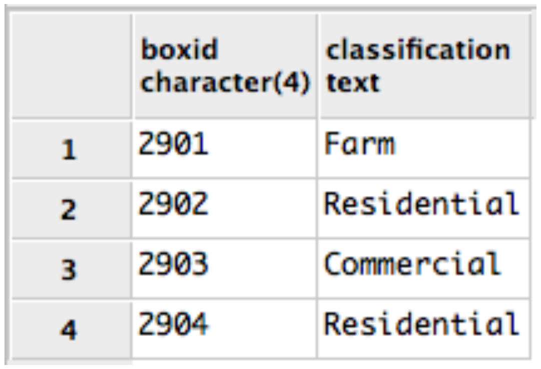
This table contains information about a box. It is related to Addresses in that many addresses can have one box. A box in firefighter is a region in a particular town. It is best understood by looking at a map and dividing a town up into different pieces, each piece being a different box. A box is used to help orchestrate what surrounding towns are called to help in an emergency.

SQL Code

```
CREATE TABLE box(
        boxID char(4) NOT NULL unique,
        classification text,
        primary key (boxID)
);
```

Functional Dependency

boxID → classification



	boxid character(4)	classification text
1	2901	Farm
2	2902	Residential
3	2903	Commercial
4	2904	Residential

Sample output from Box

MutualAid

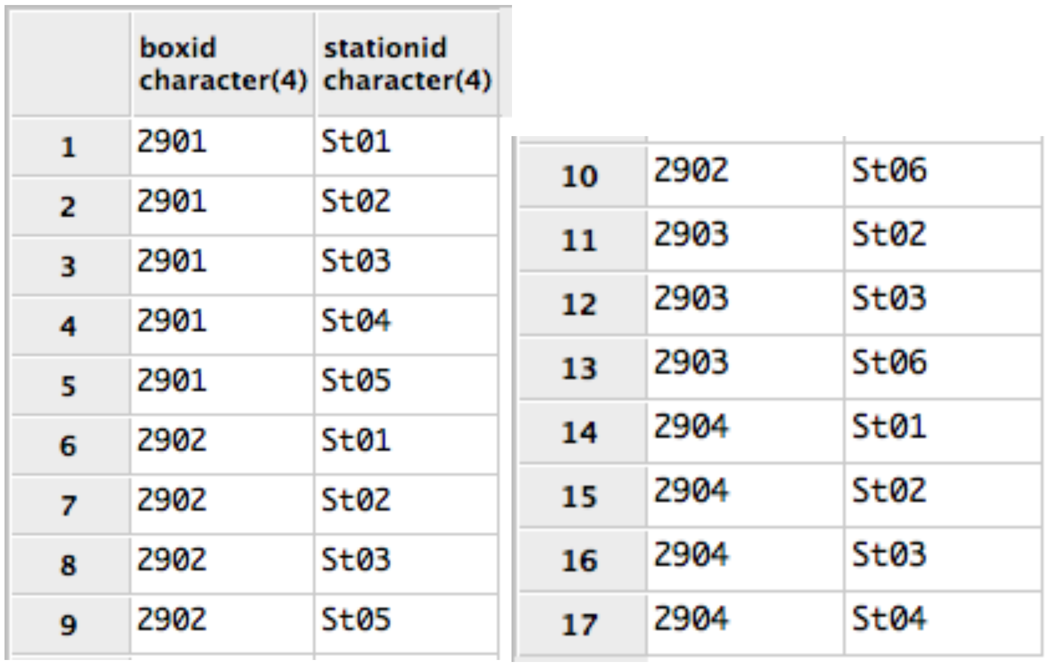
This table is an associate entity between Box and Stations.

SQL Code

```
CREATE TABLE mutualAid(
	boxID char(4) NOT NULL references box(boxID),
	stationID char(4) NOT NULL references stations(stationID),
	primary key (boxID, stationID)
);
```

Functional Dependency

<u>boxID</u>, <u>stationID</u> →



	boxid character(4)	stationid character(4)
1	2901	St01
2	2901	St02
3	2901	St03
4	2901	St04
5	2901	St05
6	2902	St01
7	2902	St02
8	2902	St03
9	2902	St05
10	2902	St06
11	2903	St02
12	2903	St03
13	2903	St06
14	2904	St01
15	2904	St02
16	2904	St03
17	2904	St04

Sample output from Box

Stations

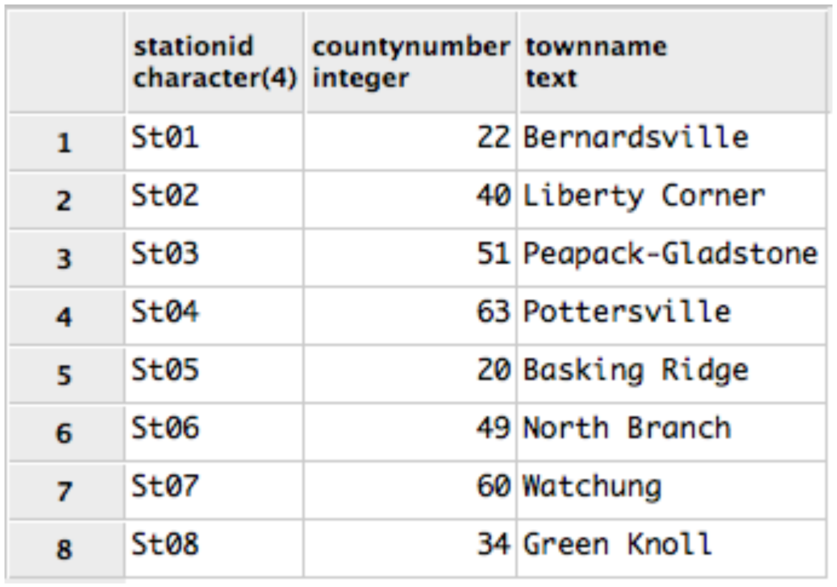
This table contains information on the mutual aid stations for Far Hills-Bedminster Fire Department. This table is connected to MutualAid and to Box. A station can be in many boxes.

SQL Code

```sql
CREATE TABLE stations(
	stationID char(4) NOT NULL,
	countyNumber int,
	townName text,
	primary key (stationID)
);
```

Functional Dependency

<u>stationID</u> → countyNumber, townName



	stationid character(4)	countynumber integer	townname text
1	St01	22	Bernardsville
2	St02	40	Liberty Corner
3	St03	51	Peapack-Gladstone
4	St04	63	Pottersville
5	St05	20	Basking Ridge
6	St06	49	North Branch
7	St07	60	Watchung
8	St08	34	Green Knoll

Sample output from Stations

ResponderApparatus

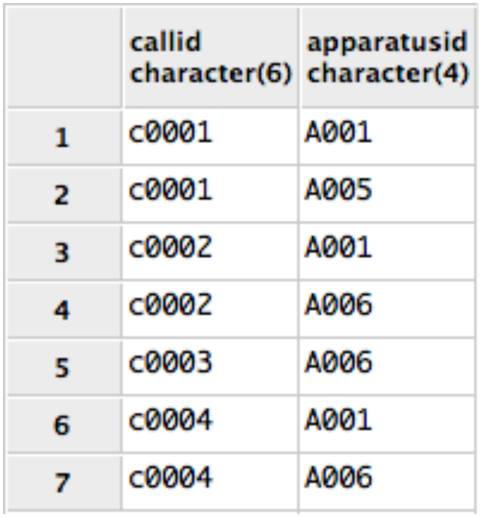
This table contains is an associate entity that connects calls and apparatus.

SQL Code

```
CREATE TABLE responderApparatus(
        callID char(6) NOT NULL references calls(callID),
        apparatusID char(4) NOT NULL references apparatus(apparatusID),
        primary key (callID, apparatusID)
);
```

Functional Dependency

<u>apparatusID</u>, <u>callID</u> →



	callid character(6)	apparatusid character(4)
1	c0001	A001
2	c0001	A005
3	c0002	A001
4	c0002	A006
5	c0003	A006
6	c0004	A001
7	c0004	A006

Sample output from ResponderApparatus

Apparatus

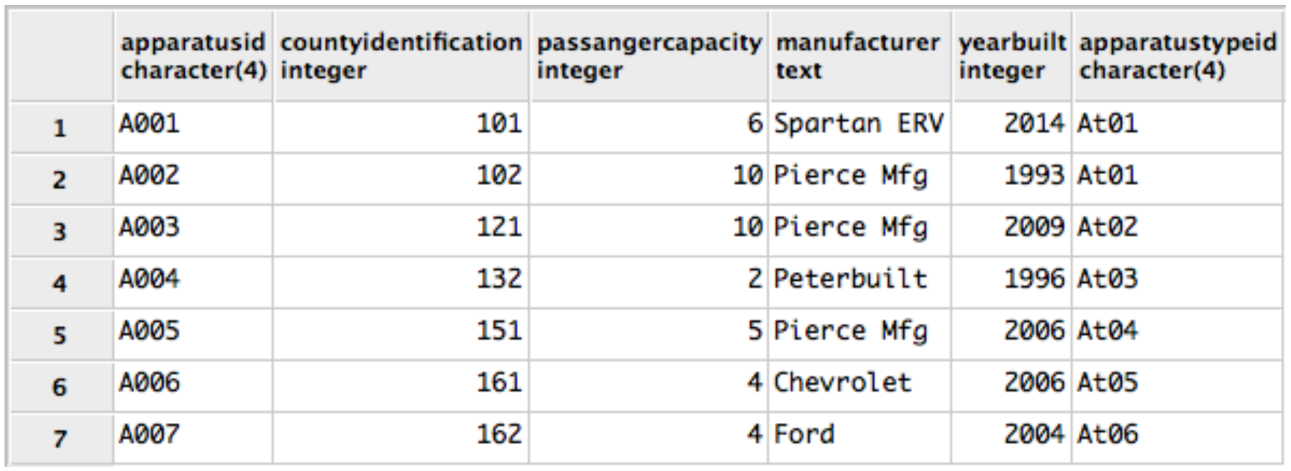
This table contains information on the apparatus that Far Hills-Bedminster has. It is related to calls through ResponderApparatus with a one-to-many relationship. Each apparatus also has an apparatus type (engine, rescue, bucket, incident command, utility), so Apparatus is also linked to ApparatusType with a one-to-one relationship.

SQL Code

```
CREATE TABLE apparatus(
    apparatusID char(4) NOT NULL unique,
    countyIdentification int,
    passangerCapacity int,
    manufacturer text,
    yearBuilt int,
    apparatusTypeID char(4) references apparatusType(apparatusTypeID),
    primary key(apparatusID)
);
```

Functional Dependency

apparatusID → countyIdentification, passangerCapacity, manufacturer, yearBuilt, apparatusTypeID



	apparatusid character(4)	countyidentification integer	passangercapacity integer	manufacturer text	yearbuilt integer	apparatustypeid character(4)
1	A001	101	6	Spartan ERV	2014	At01
2	A002	102	10	Pierce Mfg	1993	At01
3	A003	121	10	Pierce Mfg	2009	At02
4	A004	132	2	Peterbuilt	1996	At03
5	A005	151	5	Pierce Mfg	2006	At04
6	A006	161	4	Chevrolet	2006	At05
7	A007	162	4	Ford	2004	At06

Sample output from Apparatus

ApparatusType

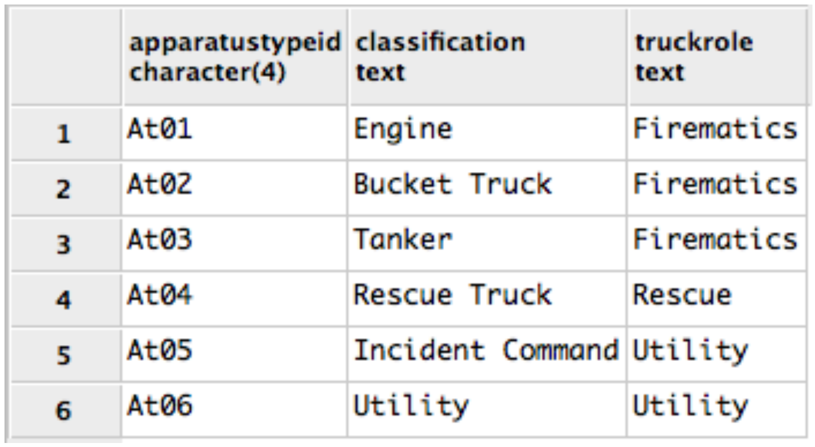
This table contains information on the different types of apparatus that Far Hills-Bedminster has. There is a one-to-many relationship to apparatus because there are many apparatuses that can be in one apparatus type.

SQL Code

```
CREATE TABLE apparatusType(
	apparatusTypeID char(4) NOT NULL unique,
	classification text,
	truckRole text,
	primary key (apparatusTypeID)
);
```

Functional Dependency

apparatusTypeID → classification, truckRole



	apparatustypeid character(4)	classification text	truckrole text
1	At01	Engine	Firematics
2	At02	Bucket Truck	Firematics
3	At03	Tanker	Firematics
4	At04	Rescue Truck	Rescue
5	At05	Incident Command	Utility
6	At06	Utility	Utility

Sample output from ApparatusType

Members

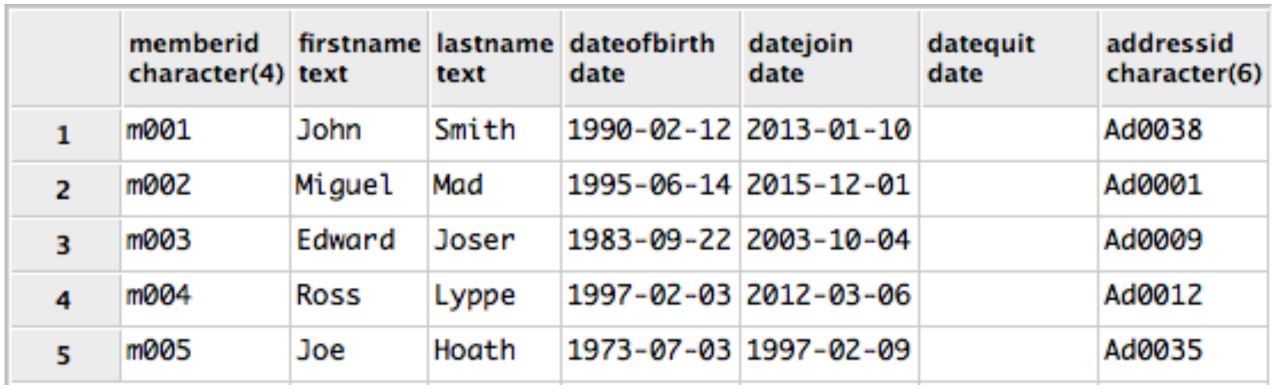
This table contains information about the members of the fire department. Each member then is also put into another table (firefighter, AssociateMember, or AdministrativeMember) in a one-to-one relationship. Members is connected to calls in a many-to-many relationship with the associate entity ResponderPeople. Members is also connected to address in a one-to-one relationship because one member has one address.

SQL Code

```
CREATE TABLE members(
        memberID char(4) NOT NULL unique,
        firstName text NOT NULL,
        lastName text NOT NULL,
        dateOfBirth date NOT NULL,
        dateJoin date check(dateJoin>dateOfBirth) NOT NULL,
        dateQuit date check(dateQuit>dateJoin),
        addressID char(6) references address(addressID),
        primary key (memberID)
);
```

Functional Dependency

memberID → firstName, lastName, dateOfBirth, dateJoin, dateQuit, addressID



	memberid character(4)	firstname text	lastname text	dateofbirth date	datejoin date	datequit date	addressid character(6)
1	m001	John	Smith	1990-02-12	2013-01-10		Ad0038
2	m002	Miguel	Mad	1995-06-14	2015-12-01		Ad0001
3	m003	Edward	Joser	1983-09-22	2003-10-04		Ad0009
4	m004	Ross	Lyppe	1997-02-03	2012-03-06		Ad0012
5	m005	Joe	Hoath	1973-07-03	1997-02-09		Ad0035

Sample output from Members

ResponderPeople

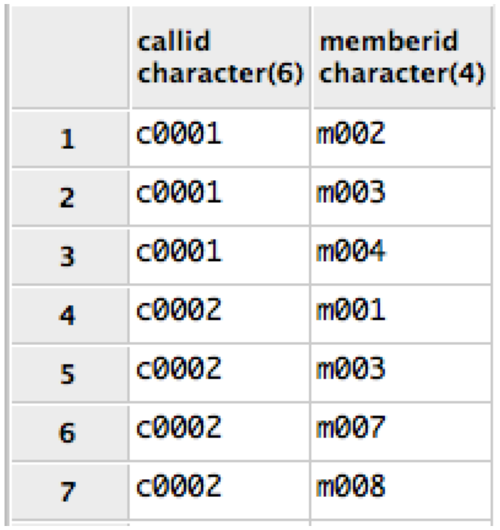
This table is an associate entity that connects Members to Calls.

SQL Code

```
CREATE TABLE responderPeople(
    callID char(6) NOT NULL,
    memberID char(4) NOT NULL,
    primary key(callID, memberID)
);
```

Functional Dependency

callID, memberID →



	callid character(6)	memberid character(4)
1	c0001	m002
2	c0001	m003
3	c0001	m004
4	c0002	m001
5	c0002	m003
6	c0002	m007
7	c0002	m008

Sample output from ResponderPeople

Firefighter

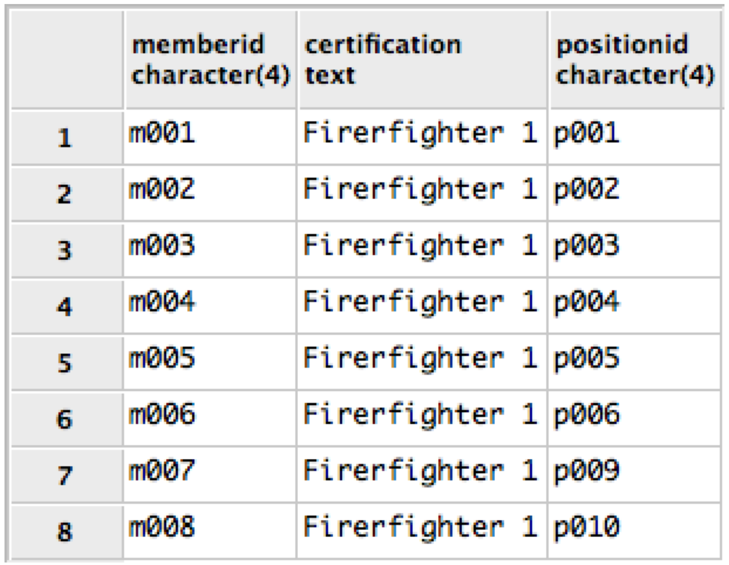
This table contains information of members who are firefighters. It is linked in a one-to-one relationship with members and a one-to-one relationship to positions.

SQL Code

```
CREATE TABLE firefighter(
        memberID char(4) NOT NULL unique references members(memberID),
        certification text,
        positionID char(4) NOT NULL references positions(positionID),
        primary key (memberID)
);
```

Functional Dependency

<u>memberID</u> → certification, positionID



	memberid character(4)	certification text	positionid character(4)
1	m001	Firerfighter 1	p001
2	m002	Firerfighter 1	p002
3	m003	Firerfighter 1	p003
4	m004	Firerfighter 1	p004
5	m005	Firerfighter 1	p005
6	m006	Firerfighter 1	p006
7	m007	Firerfighter 1	p009
8	m008	Firerfighter 1	p010

Sample output from Firefighter

AssociateMember

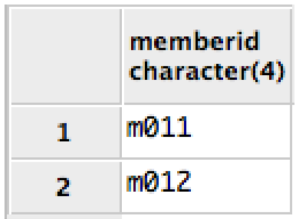
This table contains the associate members. It is linked in a one-to-one relationship with members.

SQL Code

```
CREATE TABLE associateMember(
        memberID char(4) NOT NULL unique references members(memberID),
        primary key (memberID)
);
```

Functional Dependency

<u>memberID</u> $\rightarrow$



	memberid character(4)
1	m011
2	m012

Sample output from AssociateMember

AdministrativeMember

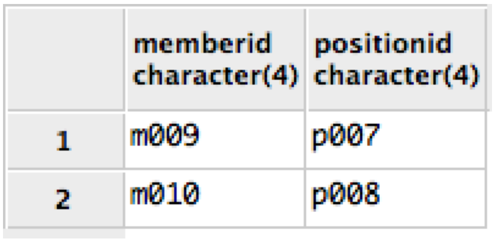
This table contains information on the administrative members in the fire department. It is linked in a one-to-one relationship with members and to positions.

SQL Code

```
CREATE TABLE administrativeMember(
        memberID char(4) NOT NULL unique references members(memberID),
        positionID char(4) NOT NULL references positions(positionID),
        primary key (memberID)
);
```

Functional Dependency

<u>memberID</u> → positionID



	memberid character(4)	positionid character(4)
1	m009	p007
2	m010	p008

Sample output from AdministrativeMember

Positions

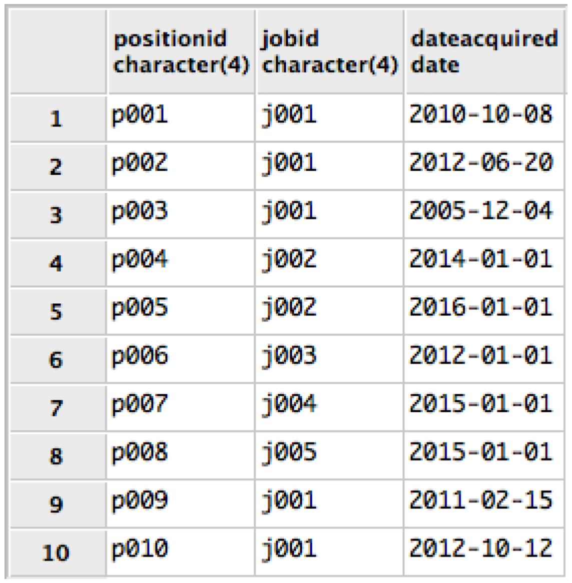
This table is an associate entity that connects Firefighter and AdministrativeMember to Job. It has a one-to-one relationship with both Firefighter and AdministrativeMember, and it has a one-to-many relationship with Job. Many positions can have one job.

SQL Code

```sql
CREATE TABLE positions(
	positionID char(4) NOT NULL unique,
	jobID char(4) NOT NULL references job(jobID),
	dateAcquired date,
	primary key (positionID)
);
```

Functional Dependency

<u>positionID</u> → jobID, dateAcquired



	positionid character(4)	jobid character(4)	dateacquired date
1	p001	j001	2010-10-08
2	p002	j001	2012-06-20
3	p003	j001	2005-12-04
4	p004	j002	2014-01-01
5	p005	j002	2016-01-01
6	p006	j003	2012-01-01
7	p007	j004	2015-01-01
8	p008	j005	2015-01-01
9	p009	j001	2011-02-15
10	p010	j001	2012-10-12

Sample output from Positions

Job

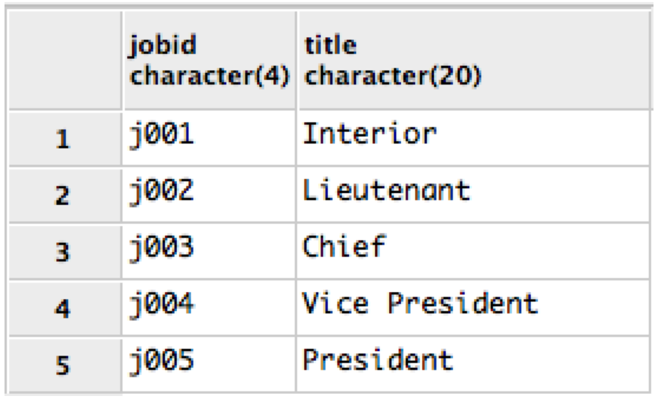
This table contains information on the different jobs in the fire department. It is connected to positions in a one-to-many relationship because many positions can have one job.

SQL Code

```sql
CREATE TABLE job(
        jobID char(4) not null unique,
        title char(20),
        primary key (jobID)
);
```

Functional Dependency

jobID → title



	jobid character(4)	title character(20)
1	j001	Interior
2	j002	Lieutenant
3	j003	Chief
4	j004	Vice President
5	j005	President

Sample output from Job

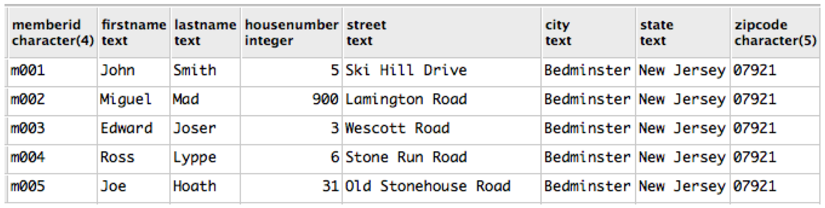
View

MemberAddress

Displays member first name, last name and full address.

SQL Code

```sql
CREATE VIEW memberAddress AS
   select members.memberID,
          members.firstName,
          members.lastName,
          address.houseNumber,
          address.street,
          address.city,
          address.state,
          address.zipcode
   from members, address
   where members.addressID = address.addressID
   order by members.memberID;
```



memberid character(4)	firstname text	lastname text	housenumber integer	street text	city text	state text	zipcode character(5)
m001	John	Smith	5	Ski Hill Drive	Bedminster	New Jersey	07921
m002	Miguel	Mad	900	Lamington Road	Bedminster	New Jersey	07921
m003	Edward	Joser	3	Wescott Road	Bedminster	New Jersey	07921
m004	Ross	Lyppe	6	Stone Run Road	Bedminster	New Jersey	07921
m005	Joe	Hoath	31	Old Stonehouse Road	Bedminster	New Jersey	07921

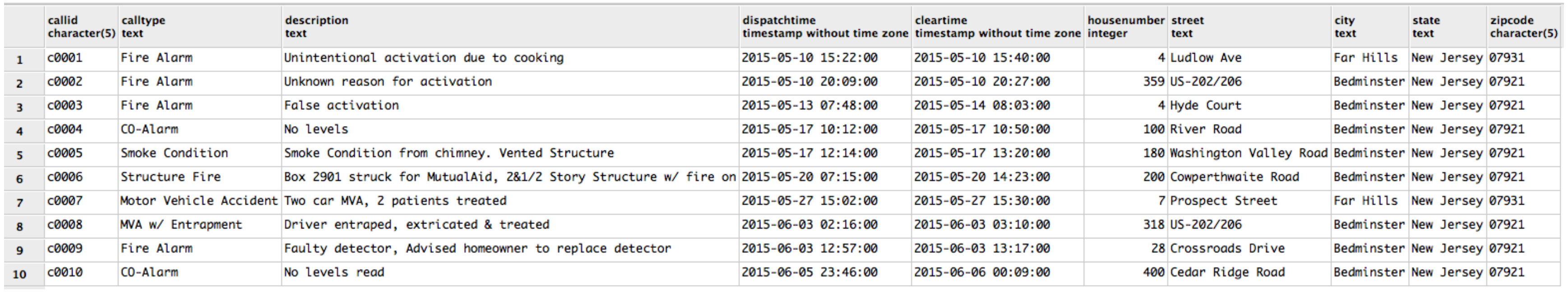
Sample output from MemberAddress

CallInformation

Displays information about a call, including the full address.

SQL Code

```sql
CREATE VIEW callInformation AS
   select calls.callID,
          calls.callType,
          calls.description,
          calls.dispatchTime,
          calls.clearTime,
          address.houseNumber,
          address.street,
          address.city,
          address.state,
          address.zipCode
   from calls, address
   where calls.addressID=address.addressID
   order by dispatchTime asc;
```



	callid character(5)	calltype text	description text	dispatchtime timestamp without time zone	cleartime timestamp without time zone	housenumber integer	street text	city text	state text	zipcode character(5)
1	c0001	Fire Alarm	Unintentional activation due to cooking	2015-05-10 15:22:00	2015-05-10 15:40:00	4	Ludlow Ave	Far Hills	New Jersey	07931
2	c0002	Fire Alarm	Unknown reason for activation	2015-05-10 20:09:00	2015-05-10 20:27:00	359	US-202/206	Bedminster	New Jersey	07921
3	c0003	Fire Alarm	False activation	2015-05-13 07:48:00	2015-05-14 08:03:00	4	Hyde Court	Bedminster	New Jersey	07921
4	c0004	CO-Alarm	No levels	2015-05-17 10:12:00	2015-05-17 10:50:00	100	River Road	Bedminster	New Jersey	07921
5	c0005	Smoke Condition	Smoke Condition from chimney. Vented Structure	2015-05-17 12:14:00	2015-05-17 13:20:00	180	Washington Valley Road	Bedminster	New Jersey	07921
6	c0006	Structure Fire	Box 2901 struck for MutualAid, 2&1/2 Story Structure w/ fire on	2015-05-20 07:15:00	2015-05-20 14:23:00	200	Cowperthwaite Road	Bedminster	New Jersey	07921
7	c0007	Motor Vehicle Accident	Two car MVA, 2 patients treated	2015-05-27 15:02:00	2015-05-27 15:30:00	7	Prospect Street	Far Hills	New Jersey	07931
8	c0008	MVA w/ Entrapment	Driver entraped, extricated & treated	2015-06-03 02:16:00	2015-06-03 03:10:00	318	US-202/206	Bedminster	New Jersey	07921
9	c0009	Fire Alarm	Faulty detector, Advised homeowner to replace detector	2015-06-03 12:57:00	2015-06-03 13:17:00	28	Crossroads Drive	Bedminster	New Jersey	07921
10	c0010	CO-Alarm	No levels read	2015-06-05 23:46:00	2015-06-06 00:09:00	400	Cedar Ridge Road	Bedminster	New Jersey	07921

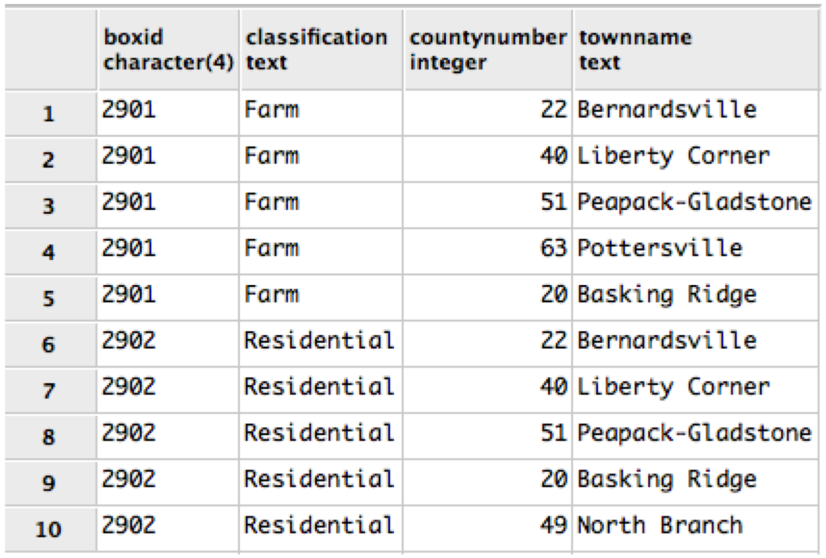
Sample output from CallInformation

BoxMutualAid

Displays all the mutual aid stations in the boxes

SQL Code

```
CREATE VIEW boxMutualAid AS
   select box.boxID,
          box.classification,
          stations.countyNumber,
          stations.townName
   from box, stations, mutualAid
   where box.boxID=mutualAid.boxID
   and   mutualAid.stationID = stations.stationID
   order by box.boxID asc;
```



	boxid character(4)	classification text	countynumber integer	townname text
1	2901	Farm	22	Bernardsville
2	2901	Farm	40	Liberty Corner
3	2901	Farm	51	Peapack-Gladstone
4	2901	Farm	63	Pottersville
5	2901	Farm	20	Basking Ridge
6	2902	Residential	22	Bernardsville
7	2902	Residential	40	Liberty Corner
8	2902	Residential	51	Peapack-Gladstone
9	2902	Residential	20	Basking Ridge
10	2902	Residential	49	North Branch

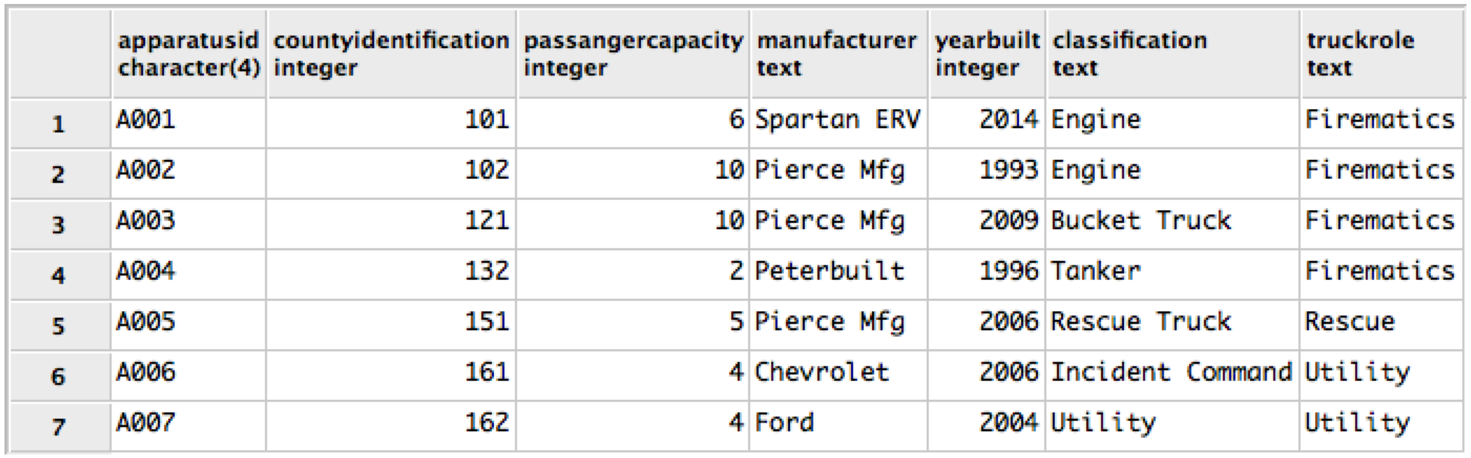
Sample output from BoxMutualAid

ApparatusInformation

Displays all the information of an apparatus, including they type and its role (firematics, rescue, utility).

SQL Code

```sql
CREATE VIEW apparatusInformation AS
    select apparatus.apparatusID,
           apparatus.countyIdentification,
           apparatus.passangerCapacity,
           apparatus.manufacturer,
           apparatus.yearBuilt,
           apparatusType.classification,
           apparatusType.truckRole
    from apparatus, apparatusType
    where apparatus.apparatusTypeID = apparatusType.apparatusTypeID;
```



	apparatusid character(4)	countyidentification integer	passangercapacity integer	manufacturer text	yearbuilt integer	classification text	truckrole text
1	A001	101	6	Spartan ERV	2014	Engine	Firematics
2	A002	102	10	Pierce Mfg	1993	Engine	Firematics
3	A003	121	10	Pierce Mfg	2009	Bucket Truck	Firematics
4	A004	132	2	Peterbuilt	1996	Tanker	Firematics
5	A005	151	5	Pierce Mfg	2006	Rescue Truck	Rescue
6	A006	161	4	Chevrolet	2006	Incident Command	Utility
7	A007	162	4	Ford	2004	Utility	Utility

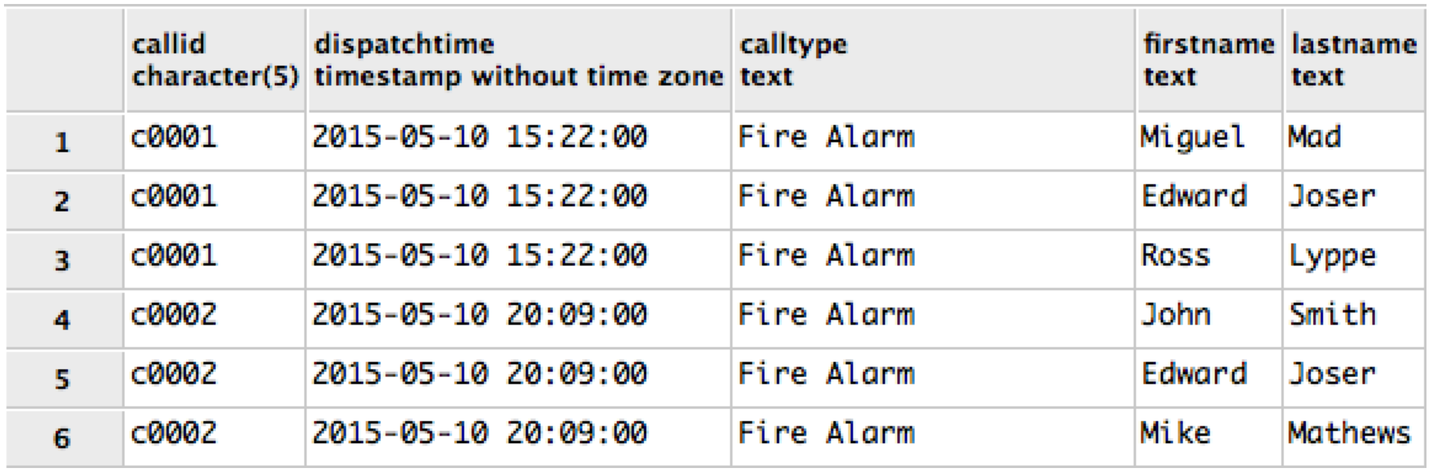
Sample output from ApparatusInformation

MembersResponding

Displays the members that respond to calls.

SQL Code

```sql
CREATE VIEW memberResponders AS
   select calls.callID,
          calls.dispatchTime,
          calls.callType,
          members.firstName,
          members.lastName
   from calls, members, responderPeople
   where calls.callID = responderPeople.callID
   and   responderPeople.memberID=members.memberID;
```



	callid character(5)	dispatchtime timestamp without time zone	calltype text	firstname text	lastname text
1	c0001	2015-05-10 15:22:00	Fire Alarm	Miguel	Mad
2	c0001	2015-05-10 15:22:00	Fire Alarm	Edward	Joser
3	c0001	2015-05-10 15:22:00	Fire Alarm	Ross	Lyppe
4	c0002	2015-05-10 20:09:00	Fire Alarm	John	Smith
5	c0002	2015-05-10 20:09:00	Fire Alarm	Edward	Joser
6	c0002	2015-05-10 20:09:00	Fire Alarm	Mike	Mathews

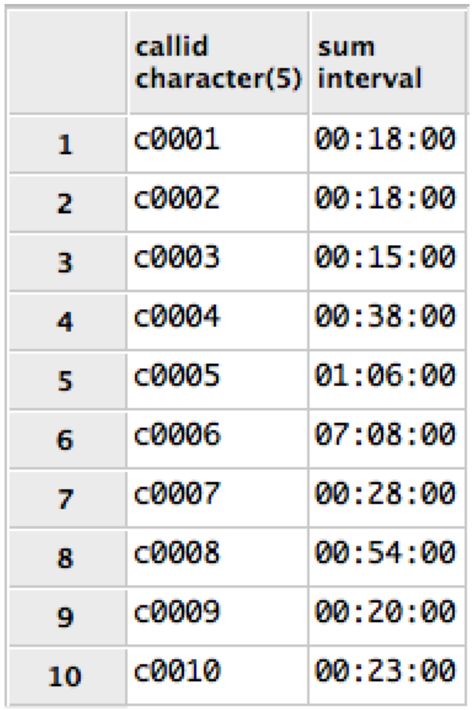
Sample output from MembersResponding

CallTime

Displays the amount of time spent on a call, used for a report to calculate the man-hours of a call.

SQL Code

```sql
CREATE VIEW callTime AS
    select callID, sum(clearTime-dispatchTime)
    from calls
    group by callID
    order by callID;
```



	callid character(5)	sum interval
1	c0001	00:18:00
2	c0002	00:18:00
3	c0003	00:15:00
4	c0004	00:38:00
5	c0005	01:06:00
6	c0006	07:08:00
7	c0007	00:28:00
8	c0008	00:54:00
9	c0009	00:20:00
10	c0010	00:23:00

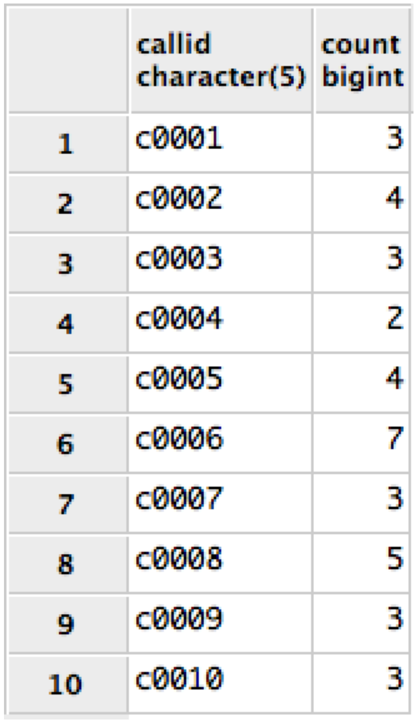
Sample output from CallTime

CallTimeResponders

Displays the number of members that responded to a call, used for a report to calculate the man-hours of a call.

SQL Code

```
CREATE VIEW callTimeResponder AS
   select calls.callID, count(members.firstName)
   from calls, members, responderPeople
   where calls.callID = responderPeople.callID
   and responderPeople.memberID=members.memberID
   group by calls.callID
   order by calls.callID;
```



	callid character(5)	count bigint
1	c0001	3
2	c0002	4
3	c0003	3
4	c0004	2
5	c0005	4
6	c0006	7
7	c0007	3
8	c0008	5
9	c0009	3
10	c0010	3

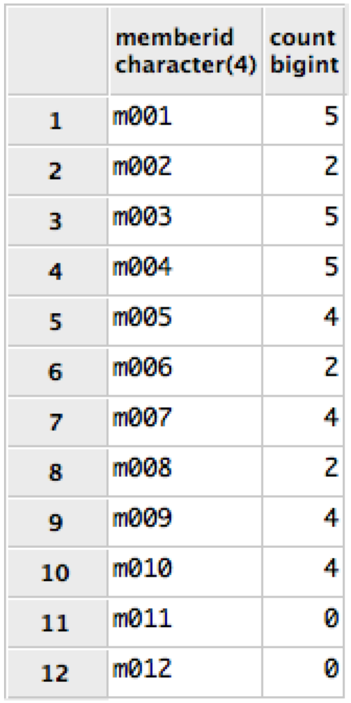
Sample output from CallTimeResponder

Reports

Query to return the amount of calls a member responded to

SQL Code

```sql
select m.memberID, count(r.callID)
from members m
full outer join responderPeople r on r.memberID = m.memberID
group by m.memberID
order by m.memberID;
```



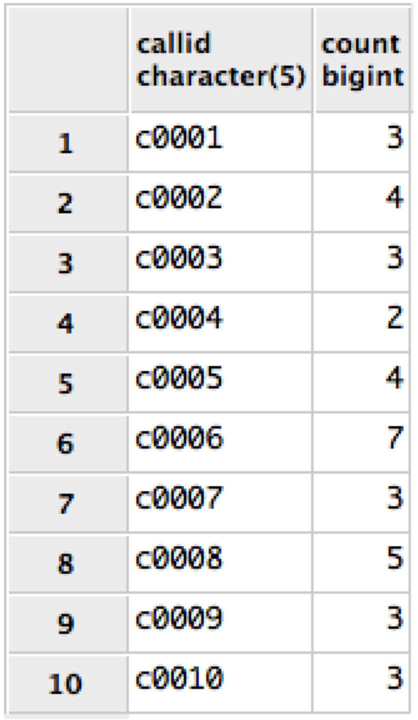
	memberid character(4)	count bigint
1	m001	5
2	m002	2
3	m003	5
4	m004	5
5	m005	4
6	m006	2
7	m007	4
8	m008	2
9	m009	4
10	m010	4
11	m011	0
12	m012	0

Sample output from query

Query to return the members that responded to a call

SQL Code

```sql
select calls.callID,
       count(members.firstName)
from calls, members, responderPeople
where calls.callID = responderPeople.callID
and   responderPeople.memberID=members.memberID
group by calls.callID
order by calls.callID;
```



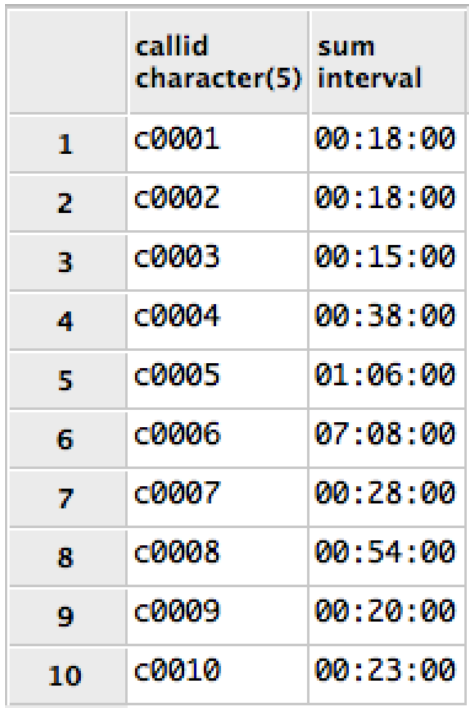
	callid character(5)	count bigint
1	c0001	3
2	c0002	4
3	c0003	3
4	c0004	2
5	c0005	4
6	c0006	7
7	c0007	3
8	c0008	5
9	c0009	3
10	c0010	3

Sample output from query

Query to return the total time spent on a call

SQL Code

```
select callID, sum(clearTime-dispatchTime)
from calls
group by callID
order by callID;
```



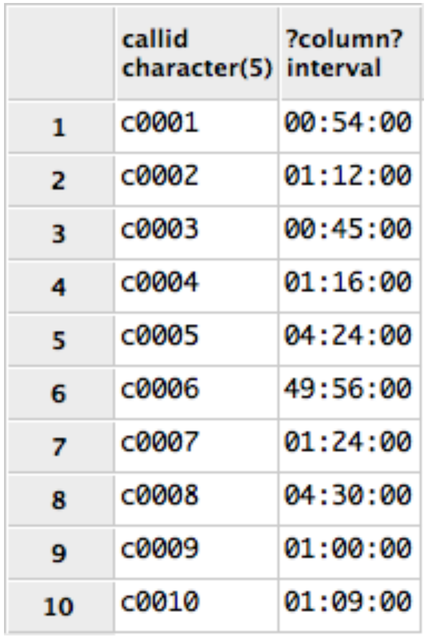
	callid character(5)	sum interval
1	c0001	00:18:00
2	c0002	00:18:00
3	c0003	00:15:00
4	c0004	00:38:00
5	c0005	01:06:00
6	c0006	07:08:00
7	c0007	00:28:00
8	c0008	00:54:00
9	c0009	00:20:00
10	c0010	00:23:00

Sample output from query

Query to return the man-hours of a call

SQL Code

```sql
select callTime.callID, callTime.sum*callTimeResponder.count
from callTime, callTimeResponder
where callTime.callID=callTimeResponder.callID
order by callTimeResponder.callID;
```



	callid character(5)	?column? interval
1	c0001	00:54:00
2	c0002	01:12:00
3	c0003	00:45:00
4	c0004	01:16:00
5	c0005	04:24:00
6	c0006	49:56:00
7	c0007	01:24:00
8	c0008	04:30:00
9	c0009	01:00:00
10	c0010	01:09:00

Sample output from query

Stored Procedures

ViewMembersResponding

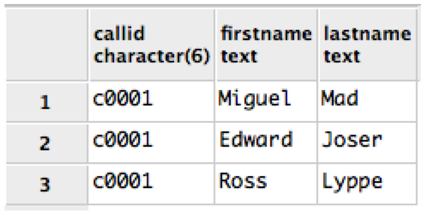
Displays the members that respond to a specific call

SQL Code

```
create function viewMembersResponding(char(5), refcursor) returns refcursor AS
$$
declare
   callInput text := $1;
   resultSet refcursor := $2;
begin
   open resultSet for
      select responderPeople.callID,
             members.firstName,
             members.lastName
      from members, responderPeople
      where callInput= responderPeople.callID
      and   responderPeople.memberID=members.memberID;
   return resultSet;
end;
$$
language plpgsql;
```

Select statement

```
select viewMembersResponding('c0001', 'results');
fetch all from results;
```



	callid character(6)	firstname text	lastname text
1	c0001	Miguel	Mad
2	c0001	Edward	Joser
3	c0001	Ross	Lyppe

Sample output from ViewMembersResponding

ViewMemberInformation

Displays the member information, including their job (if applicable)

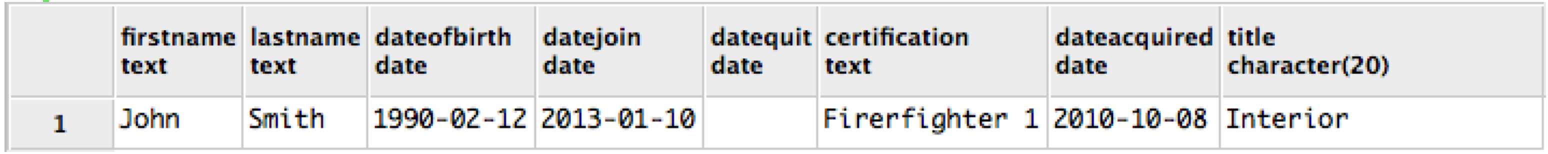
SQL Code

```sql
create function viewMemberInformation(char(4), refcursor) returns refcursor AS
$$
declare
   memberIDInput text := $1;
   resultSet refcursor := $2;
begin
      if exists (select memberID from firefighter where memberID=memberIDInput) then
            open resultSet for
               select members.firstName, members.lastName, members.dateOfBirth, members.dateJoin, members.dateQuit,
firefighter.certification, positions.dateAcquired job.title
               from members, firefighter, positions, job
               where members.memberID=memberIDInput
               and members.memberID=firefighter.memberID
               and firefighter.positionID=positions.positionID
               and positions.jobID=job.jobID;
            return resultSet;
      end if;
      if exists (select memberID from administrativeMember where memberID=memberIDInput) then
            open resultSet for
               select members.firstName, members.lastName, members.dateOfBirth, members.dateJoin, members.dateQuit,
 positions.dateAcquired, job.title
               from members, positions, job, administrativeMember
               where members.memberID=memberIDInput
               and members.memberID=administrativeMember.memberID
               and administrativeMember.positionID=positions.positionID
               and positions.jobID=job.jobId;
            return resultSet;
      end if;
      if exists (select memberID from associateMember where memberID=memberIDInput) then
            open resultSet for
               select members.firstName, members.lastName, members.dateOfBirth, members.dateJoin, members.dateQuit
               from members, associateMember
               where members.memberID=memberIDInput
               and members.memberID=associateMember.memberID;
            return resultSet;
      end if;
end;
```

```
$$
language plpgsql;
```

Select statement

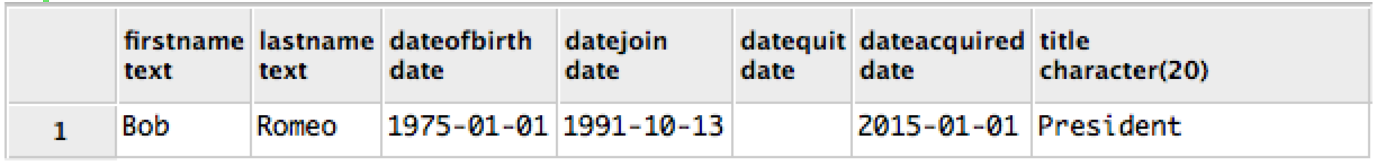
```
select viewMemberInformation('m001', 'results');
fetch all from results;
```



	firstname text	lastname text	dateofbirth date	datejoin date	datequit date	certification text	dateacquired date	title character(20)
1	John	Smith	1990-02-12	2013-01-10		Firerfighter 1	2010-10-08	Interior

Select statement

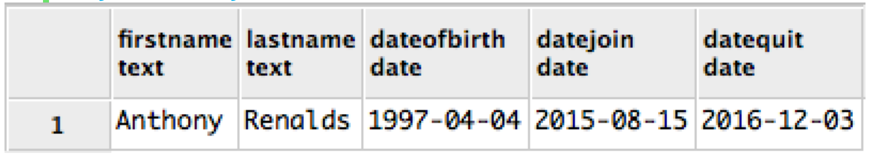
```
select viewMemberInformation('m010', 'results');
fetch all from results;
```



	firstname text	lastname text	dateofbirth date	datejoin date	datequit date	dateacquired date	title character(20)
1	Bob	Romeo	1975-01-01	1991-10-13		2015-01-01	President

Select statement

```
select viewMemberInformation('m012', 'results');
fetch all from results;
```



	firstname text	lastname text	dateofbirth date	datejoin date	datequit date
1	Anthony	Renalds	1997-04-04	2015-08-15	2016-12-03

ViewMutualAid

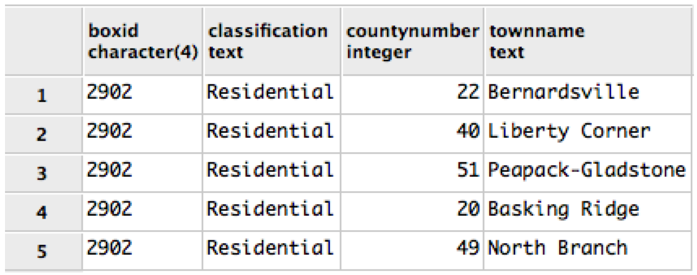
Displays the mutual aid stations for a box

SQL Code

```
create function viewMutualAid(char(4), refcursor) returns refcursor AS
$$
declare
   boxInput text := $1;
   resultSet refcursor := $2;
begin
   open resultSet for
      select box.boxID,
          box.classification,
          stations.countyNumber,
          stations.townName
      from box, stations, mutualAid
      where boxInput=box.boxID
      and box.boxID=mutualAid.boxID
      and   mutualAid.stationID = stations.stationID;
   return resultSet;
end;
$$
language plpgsql;
```

Select statement

```
select viewMutualAid(‘2902’, 'results');
fetch all from results;
```



	boxid character(4)	classification text	countynumber integer	townname text
1	2902	Residential	22	Bernardsville
2	2902	Residential	40	Liberty Corner
3	2902	Residential	51	Peapack-Gladstone
4	2902	Residential	20	Basking Ridge
5	2902	Residential	49	North Branch

Sample output from ViewMutualAid

InsertIntoFirefighter (InsertIntoAssociateMember & InsertIntoAdministrativeMember)

Checks to see if a memberID is in the other member subtype tables

SQL Code

```sql
create function insertIntoFirefighter(char(4)) returns boolean AS
$$
declare
   memberInput text := $1;
begin
   if exists (select memberID from associateMember where memberID=memberInput) then
      return true;
   elsif exists (select memberID from administrativeMember where memberID=memberInput) then
      return true;
   else
      return false;
   end if;
end;
$$
language plpgsql;
```

This function also has variants insertIntoAssociateMember and insertIntoAdministrativeMember.

NewMember()

Trigger to see if a memberID is in a member subtype table

SQL Code

```
create function newMember() returns trigger AS
$$
declare
   test1 boolean = insertIntoFirefighter(new.memberID);
   test2 boolean = insertIntoAssociateMember(new.memberID);
   test3 boolean = insertIntoAdministrativeMember(new.memberID);
begin
   if(test1=true) then
      raise exception 'Member is already in another table';
   elsif (test2=true) then
      raise exception 'Member is already in another table';
   elsif(test3=true) then
      raise exception 'Member is already in another table';
   else
      return new;
   end if;
end;
$$
language plpgsql;
```

Triggers

NewMember

This trigger runs before an insert on the subtype tables Firefighter, AssociateMember, and AdministrativeMember to make sure that the MemberID being inserted is not already in one of the other two tables. This trigger uses the function newMember(). newMember() uses functions insertIntoFirefighter, insertIntoAssociateMember, and insertIntoAdministrativeMember. This trigger is used three times (the three subtype tables).

```
create trigger newMember
before insert on firefighter
for each row
execute procedure newMember();
```

```
create trigger newMember
before insert on associateMember
for each row
execute procedure newMember();
```

```
create trigger newMember
before insert on administrativeMember
for each row
execute procedure newMember();
```

Security

There are four different roles in the database that users fall under. Users can be an Admin, Officer, Member, or Administrative Member. Depicted below are each one.

Admin

An Admin has complete control over the database. He or she can edit any part of the database.

SQL Code

```
create role admin;
grant all on all tables in schema public to admin;
```

Officer

An officer is a member who is a lieutenant or chief. He or she has access to selecting information about calls, people who responded, apparatuses that responded, addresses, boxes, mutual aid, and stations. He or she can also insert new calls and its pertinent information (addresses, members & apparatus that responded). Finally, he or she can update information on previous calls and its pertinent information.

SQL Code

```sql
create role officer;
grant select on calls, responderPeople, responderApparatus,
            apparatus, apparatusType, address, box, mutualAid,
            stations
to officer;
grant insert on calls,
            responderPeople,
            responderApparatus,
            address
to officer;
grant update on calls,
            responderPeople,
            responderApparatus,
            address
to officer;
```

Member

A member can select information about members and their positions, as well as addresses. Members can update information about members and addresses, but he or she cannot insert or delete any information.

SQL Code

```
create role members;
grant select on members,
            firefighter,
            associateMember,
            administrativeMember,
            positions,
            job,
            address
to members;
grant update on members, address
to members;
```

AdministrativeMember

An administrative member is a member who is either the president or vice president. He or she have the same ability as members, but they also can insert new positions and jobs. He or she can also update information on current members and current positions and jobs.

SQL Code

```
Create role administrativeMember;
grant select on members,
            firefighter,
            associateMember,
            administrativeMember,
            positions,
            job,
            address
to administrativeMember;
grant insert on positions,
          job
to administrativeMember;
grant update on members,
            positions,
            job
to administrativeMember;
```

Implementation Notes

As noted below, there was a problem when implementing my store procedures. In each store procedure definition, I had the select and fetch statements below and it would not let me implement them all at once, only individually. The solution to this was to comment out the select and fetch statements. The Full Database File.sql file does not have comments in it (to make it as condense as possible), but the individual files for the implementation, insert data, etc. all have comments in them. Additionally, as mentioned below, the '*drop role*' syntax does not function correctly, but it is included in the .sql file.

Additionally, (as mentioned), all the information in this database is fictional. This database is property of Robert Lynch. All rights are reserved.

Known Problems

One of the biggest problems is the when implementing the database, the store procedures cannot be implemented with their select statements, they must be implemented and then queried. I am not quite sure why, but I have done some research to no avail. Another problem is that members are only allowed to have one position, but it the real application, members can have multiple positions. I cannot currently drop a role with the '*drop role*' syntax, instead, I must go into PGAdmin and delete it using the GUI.

Future Enhancements

Some future enhancements would be to add more members, calls, addresses, etc., to diversify the database even more. I would also like to add the ability for a member to have more than one position (i.e. president and lieutenant), which does happen the real application. Another enhancement I would like to make is the ability for a box alarm. Box alarms use boxes to decide what mutual aid companies to dispatch. For example, a 1st alarm would dispatch one company from box list and a 2nd alarm would dispatch two companies from the box list. This also doesn't contain all the information needed to run a real fire department database because there are additional things like apparatus maintenance, member gear, tools, etc. to have to be included.