



## A Database Design Proposal for Marist College ResNet

**By Gabrielle Romano**

December 2<sup>nd</sup> 2013

## Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>Executive Summary .....</b>	<b>3</b>
<b>Entity Relationship Diagram .....</b>	<b>4</b>
<b>Create Table Statements</b>	
<i>People</i> table .....	5
<i>Employees</i> table .....	6
<i>Employee_Payments</i> table .....	7
<i>Customer</i> table .....	8
<i>Customers_Request</i> table .....	9
<i>Services</i> table .....	10
<i>Customers_Process</i> table .....	11
<i>Customers_Status</i> table .....	12
<i>Status</i> table .....	13
<i>Devices</i> table .....	14
<i>Device_Types</i> table .....	15
<i>Types</i> table .....	16
<b>Views .....</b>	<b>17</b>
<b>Reports .....</b>	<b>19</b>
<b>Stored Procedures .....</b>	<b>21</b>
<b>Triggers .....</b>	<b>23</b>
<b>Security .....</b>	<b>24</b>
<b>Known Problems/Future Enhancements .....</b>	<b>25</b>



## Executive Summary

The Goal of this database proposal is to support Marist College Residential Networking computer services operations. Currently ResNet is supporting 6,000 combined students and faculty through various technological services; this database documents the employee's actions and customer's devices that have had technical difficulties as well as the status of a device. The database will contain information about employees and customers. It will also store records of customer's devices as well as the types in order to generate statistics about problems with specific devices. This will allow other departments to analyze the issue and provide specific solutions to deal with large scale problems.

The first section of this proposal displays the layout of the database design and the relationships between each of the tables within the database.

The following section of the proposal displays and describes an in-depth description of each table within the layout of the database. This includes a description of each table, its functional dependencies, create statements and sample data.

The third section of this proposal shows examples of views and reports that would be useful in the implementation of this database.

The fourth section documents simple possible triggers and stored procedures that can be implemented within the database.

The Last segment will provide a few final notes surrounding the project as well as some ideas to consider as we look towards the future.



## Entity Relationship Diagram



## Create Statements

### ***People table:***

Employees and Customers can be people and will inherit a person's traits from the People table. CWID (Campus Wide ID) is a unique number distributed to each student and faculty on campus.

### **Functional Dependencies:**

**CWID** → First\_Name, Last\_Name, Phone\_Number, Email, PGroup

### **Create Statement:**

CREATE TABLE People (

```

    CWID            INT            NOT NULL        UNIQUE        PRIMARY KEY,
    First_Name      VARCHAR(25)    NOT NULL,
    Last_Name       VARCHAR (25)   NOT NULL,
    Phone_Number    INT            NOT NULL,
    Email           VARCHAR (50),
    PGroup          VARCHAR (50)

```

);

### **Sample Data:**

CWID	First_Name	Last_Name	Phone_Number	Email	PGroup
20084734	Gabrielle	Romano	845-747-9874	Gabrielle.Romano@marist.edu	Student
20034852	Travis	Beatty	845-834-1273	Travis.Beatty@marist.edu	Student
20074623	Christopher	Phrisce	845-832-9264	Christopher.Phrisce@marist.edu	Student
10198273	Jimmy	Chan	845-982-7463	Jimmy.Chan@marist.edu	Student
20093847	TK	Black	845-726-9182	TK.Black@marist.edu	Student
10192836	Ricky	Adams	845-736-9283	Ricky.Adams@marist.edu	Faculty
20043192	John	Smith	845-234-5433	John.Smith@marist.edu	Student
10124234	Amy	Baker	845-856-9483	Amy.Baker@marist.edu	Student
10194739	James	Jordan	201-453-8754	James.Jordan@marist.edu	Faculty
10192837	Jack	Grand	845-847-7362	Jack.Grand@marist.edu	Student
10198726	Jessica	Taylor	201-532-6251	Jessica.Taylor@marist.edu	Faculty
20098726	Tylor	Laoufter	201-918-8373	Tylor.Laoufter@marist.edu	Student

## Create Statements

### *Employees table:*

The Employees table is a sub-type of the People table in order to keep track of all basic employee information. The added information within the Employees table is exclusive to each individual Employee.

### Functional Dependencies:

**CWID** → Stid, Hire\_Date

### Create Statement:

```
CREATE TABLE Employees (
    CWID          INT          NOT NULL    UNIQUE    REFERENCES People(CWID),
    Stid          VARCHAR(50)  NOT NULL,
    Hire_Date     DATE,
    PRIMARY KEY ( CWID )
);
```

### Sample Data:

CWID	Stid	Hire_Date
20084734	strg1	2012-05-10
20034852	stbe1	2011-05-20
20074623	stpp3	2011-11-18
10198273	stjc1	2010-08-10
20093847	sttk2	2013-09-19
10192836	stra2	2011-12-15

## Create Statement

### ***Employee\_Payments table:***

The Employee\_Payments table displays employees' salaries and the days in which they are paid based off of the CWID and Paymentid.

### **Functional Dependencies:**

**Paymentid, CWID → SalaryUSD, Payment\_Date**

### **Create Statement:**

```
CREATE TABLE Employee_Payments (
    Paymentid          INT    NOT NULL,
    CWID               INT    NOT NULL    REFERENCES People(CWID),
    SalaryUSD          INT    NOT NULL,
    Payment_Date       DATE   NOT NULL,
    PRIMARY KEY ( Paymentid, CWID )
);
```

### **Sample Data:**

Paymentid	CWID	SalaryUSD	Payment_Date
1	20084734	8.00	2013-11-30
2	20034852	9.00	2013-10-30
3	20074623	8.00	2013-11-15
4	10198273	10.00	2013-10-15
5	20093847	9.00	2013-08-15
6	10192836	40.00	2013-11-30
7	20084734	8.00	2013-12-15
8	20034852	9.00	2013-11-15
9	20074623	8.00	2013-11-30
10	10198273	9.00	2013-10-30
11	20093847	10.00	2013-09-15
12	10192836	40.00	2013-12-15

## Create Statements

### ***Customers table:***

This table records the CWID of Customers who have dropped of their device at ResNet to be serviced.

### **Functional Dependencies:**

**CWID →**

### **Create Statement:**

```
CREATE TABLE Customers (  
    CWID          INT      NOT NULL      UNIQUE      REFERENCES People(CWID),  
    PRIMARY KEY ( CWID )  
);
```

### **Sample Data:**

CWID
20043192
10124234
10194739
10192837
10198726
20098726



## Create Statements

### ***Customers\_Request*** table:

The Customers\_Request table displays a list of all the CWID's (people) who have had their computer serviced at ResNet by relating it to a unique Processid. This table also shows what services a person requested to be applied on their device.

### **Functional Dependencies:**

**Processid, CWID → Servicesid, Date**

### **Create Statement:**

```
CREATE TABLE Customers_Request (
    Processid    INT    NOT NULL,
    CWID         INT    NOT NULL    REFERENCES People(CWID),
    Servicesid   INT    NOT NULL    REFERENCES Services(Servicesid),
    CDate        DATE   NOT NULL,
    PRIMARY KEY ( Processid, CWID )
);
```

### **Sample Data:**

Processid	CWID	Servicesid	CDate
1	20043192	1	2013-10-12
2	20043192	2	2013-10-30
3	20043192	6	2013-11-27
4	20043192	2	2013-12-03
5	10124234	9	2012-08-12
6	10124234	8	2013-10-30
7	10194739	10	2011-11-27
8	10194739	6	2013-12-03
9	10194739	3	2011-11-27
10	10194739	1	2013-12-03
11	10192837	1	2010-04-02
12	10198726	2	2011-11-27
13	10198726	4	2013-12-03
14	10198726	1	2010-04-02
15	20098726	8	2010-06-02

## Create Statements

### ***Services table:***

The Services table illustrates a list of the possible services employees can perform on the various devices that customers bring in to ResNet.

### **Functional Dependencies:**

**Servicesid** → SName

### **Create Statement:**

```
CREATE TABLE Services (
    Servicesid    INT          NOT NULL    PRIMARY KEY,
    SName         VARCHAR(25) NOT NULL
);
```

### **Sample Data:**

Servicesid	SName
1	Virus Scan
2	Network Setup
3	Hardware Repair
4	OS Updates
5	Mail Setup
6	Network Exception
7	Reimage
8	Secure Data
9	Port Activation
10	Reinstall OS

## Create Statements

### ***Customer\_Process table:***

The Customer\_Process table shows what process corresponds with a device. The composite primary key of the processeid and computerid allows for consistency.

### **Functional Dependencies:**

**Processid, Computerid →**

### **Create Statement:**

```
CREATE TABLE Customers_Process (
    Processid      INT          NOT NULL      REFERENCES Customers_Requests(Processid),
    Computerid     VARCHAR(50)  NOT NULL      REFERENCES Devices(Computerid),
    PRIMARY KEY ( Processid, Computerid )
);
```

### **Sample Data:**

Processid	ComputerID
1	00:13:00:E1:11:11
2	00:13:00:E1:11:11
3	00:12:11:E2:12:12
4	00:13:33:E3:33:44
5	00:0C:29:4D:71:EB
6	00:0B:30:5D:72:EA
7	00:00:5A:99:62:50
8	00:00:5A:99:62:50
9	00:00:5B:77:63:40
10	00:99:5C:44:62:41
11	00:99:12:A2:22:3F
12	00:87:F3:A2:98:BA
13	00:87:F3:A2:98:BA
14	00:89:F4:A3:99:AB
15	00:34:A5:7D:F3:60

## Create Statements

### ***Customers\_Status table:***

The Customers\_Status table illustrates a list of processes and corresponding Statusid's to determine what status a customer's device is in.

### **Functional Dependencies:**

**Processid, Statusid →**

### **Create Statement:**

```
CREATE TABLE Customers_Status (
    Processid          INT    NOT NULL    REFERENCES Customers_Process(Processid),
    Statusid           INT    NOT NULL    REFERENCES Status(Statusid),
    PRIMARY KEY ( Processid, Statusid )
);
```

### **Sample Data:**

Processid	Statusid
1	1
2	2
3	1
4	4
5	1
6	3
7	3
8	1
9	1
10	1
11	1
12	1
13	3
14	1
15	4

## Create Statements

### ***Status table:***

The Status table illustrates a list of the possible status conditions of a computer or other device that was or is in the process of being repaired.

### **Functional Dependencies:**

**Statusid** → Status

### **Create Statement:**

```
CREATE TABLE Status (  
    Statusid INT NOT NULL PRIMARY KEY,  
    Status VARCHAR(50)  
);
```

### **Sample Data:**

Statusid	Status
1	Fixed
2	Hardware Repair
3	Not Fixed
4	In Progress
5	Picked Up

## Create Statements

### ***Devices table:***

The Devices table illustrates a list of computers and mobile devices that were brought into ResNet for repair as well as their corresponding information.

### **Functional Dependencies:**

**Computerid** → Brand, Model, OS, Password

### **Create Statement:**

```
CREATE TABLE Devices (
    Computerid    VARCHAR(50) NOT NULL PRIMARY KEY,
    Brand         VARCHAR(50),
    Model         VARCHAR(50),
    OS            VARCHAR(50),
    Password      VARCHAR(50)
);
```

### **Sample Data:**

Computerid	Brand	Model	OS	Password
00:13:00:E1:11:11	Apple	MacBookPro	10.9	Baseball1
00:12:11:E2:12:12	Apple	IPhone	6.8	9483
00:13:33:E3:33:44	Microsoft	Null	Null	Null
00:0C:29:4D:71:EB	Lenovo	T510	Windows 7	Iluvmykitty
00:0B:30:5D:72:EA	Andriod	Galexyl SIII	Null	Null
00:00:5A:99:62:50	Apple	IMac	8.3	Snapcracklepop
00:00:5B:77:63:40	Sony	Null	Null	Null
00:99:5C:44:62:41	Apple	IPad 2	7.9	9876
00:99:12:A2:22:3F	Lenovo	T530	Windows 8	Grass22
00:87:F3:A2:98:BA	Toshiba	Null	Windows 8	BlackFriday
00:89:F4:A3:99:AB	Nintendo	Null	Null	Null
00:34:A5:7D:F3:60	Microsoft	Surface	Windows 8	847394Gs

## Create Statements

### ***Device\_Types table:***

The Device\_Types table illustrates the correlation of a list of devices to its corresponding device type.

### **Functional Dependencies:**

**Computerid, Typid →**

### **Create Statement:**

```
CREATE TABLE Device_Types (
    Computerid    VARCHAR(50) NOT NULL REFERENCES Devices(Computerid),
    Typid         INT         NOT NULL REFERENCES Types(Typid),
    PRIMARY KEY ( Computerid, Typid )
);
```

### **Sample Data:**

Computerid	Typid
00:13:00:E1:11:11	1
00:12:11:E2:12:12	3
00:13:33:E3:33:44	8
00:0C:29:4D:71:EB	1
00:0B:30:5D:72:EA	3
00:00:5A:99:62:50	2
00:00:5B:77:63:40	9
00:99:5C:44:62:41	4
00:99:12:A2:22:3F	1
00:87:F3:A2:98:BA	1
00:89:F4:A3:99:AB	10
00:34:A5:7D:F3:60	4

## Create Statements

### *Types table:*

The Types table illustrates a list of the possible types a device can be considered. Views on this table could be useful for determining statistics about how many of a certain type of device is being serviced

### Functional Dependencies:

**Typeid** → Computer\_Type

### Create Statements:

```
CREATE TABLE Types (
    Typeid          INT      NOT NULL      UNIQUE      REFERENCES Types(Typeid),
    Computer_Type   VARCHAR(50) NOT NULL,
    PRIMARY KEY ( Typeid )
);
```

### Sample Data:

Typeid	Computer_Type
1	Laptop
2	Desktop
3	Phone
4	Tablet
5	Server
6	Printer
7	Switch
8	Xbox
9	PlayStation
10	Wii



## Views

### ***Views: Device\_Numbers***

This view is used to calculate how many times a particular brand of device was serviced at ResNet. This could be useful for gathering statistics on devices that come in multiple times with the same problems. This example shows how many Laptops come in and what service they have been provided with. Similar views can be created by altering the sub query where clause with a different type of device.

#### **Create View Query:**

```
CREATE VIEW Device_Numbers AS

    SELECT count(d.computerid), d.brand, s.sname
    FROM devices d
    INNER JOIN Device_Types dt ON d.Computerid = dt.Computerid
    INNER JOIN Customers_Process cp ON d.Computerid = cp.Computerid
    INNER JOIN Customers_Requests cr ON cp.Processid = cr.Processid
    INNER JOIN Services s ON cr.Servicesid = s.Servicesid
    WHERE typeid in (
        SELECT t.Typeid
        FROM Types t
        WHERE Computer_Type = 'Laptop'
    )
GROUP BY d.brand, s.SName group by d.brand
ORDER BY d.brand
```

## Views

***Views: Customers\_Services***

This view is used to calculate what services have been done on a device. This could be useful for figuring out what Services have been done on a device in the past in order to prepare solutions for future problems.

**Create View Query:**

```
CREATE VIEW Customers_Services AS
```

```
    SELECT count(s.servicesid), s.Sname, cr.CWID, p.First_Name, p.Last_Name
    FROM Customers_Requests cr
    INNER JOIN People p  ON cr.cwid = p.cwid
    INNER JOIN services s  ON cr.servicesid = s.servicesid
    GROUP BY cr.cwid, p.first_name, p.last_name,s.sname
    ORDER BY p.First_Name DESC
```

**Sample Data:**

Count	SName	CWID	First_Name
1	Port Activation	10124234	Amy
1	Secure Data	10124234	Amy
1	Virus Scan	10192837	Jack
1	Hardware Repair	10194739	James
1	Network Exception	10194739	James
1	Reinstall OS	10194739	James
1	Virus Scan	10194739	James
1	Network Setup	10198726	Jessica
1	OS Updates	10198726	Jessica
1	Virus Scan	10198726	Jessica
1	Network Exception	20043192	John
2	Network Setup	20043192	John
1	Virus Scan	20043192	John
1	Secure Data	20098726	Tylor

## Reports

### ***Reports:***

This report shows the students or faculty who have had Virus Scans on their devices. This could also be used on all of the other services if specified in the last sub-query.

### **Report Query:**

```
SELECT First_Name, Last_Name, CWID
FROM People
WHERE CWID IN (
    SELECT CWID
    FROM Customers
    WHERE CWID IN (
        SELECT CWID
        FROM Customers_Requests
        WHERE Servicesid IN (
            SELECT Servicesid
            FROM services
            WHERE Sname = 'Virus Scan'
        )
    )
)
ORDER BY Last_Name ASC
);
```

### **Sample Data:**

First_Name	Last_Name	CWID
John	Smith	20043192
James	Jordan	10194739
Jack	Grand	10192837
Jessica	Taylor	10198726

## Reports

### ***Reports:***

This report shows how many times a customer has brought in a device to be serviced.

### **Report Query:**

```
SELECT count(c.CWID),
       c.CWID,
       p.First_Name as "First Name",
       p.Last_Name as "Last Name"
FROM Customers_Requests c
INNER JOIN People p
      ON c.CWID = p.CWID
GROUP BY c.CWID, p.First_Name, p.Last_Name
ORDER BY count(c.CWID) DESC
```

### **Sample Data:**

Count	CWID	First Name	Last Name
4	20043192	John	Smith
4	10194739	James	Jordan
3	10198726	Jessica	Taylor
2	10124234	Amy	Baker
1	10192837	Jack	Grand
1	20098726	Taylor	Laoufter

## Stored Procedures

### ***Stored Procedures:***

This stored procedure is used to find out how many devices are in the office are currently in the process of being repaired.

#### **Function devices\_in\_progress:**

```
CREATE OR REPLACE FUNCTION devices_in_progress(Int)
```

```
RETURNS integer AS $$
```

```
DECLARE
```

```
runs integer;
```

```
BEGIN
```

```
    SELECT cr.CWID
```

```
    FROM customers_requests cr
```

```
    INNER JOIN Customers_Status cs
```

```
        ON cr.Processid = cs.Processid
```

```
    INNER JOIN Status s
```

```
        ON cs.statusid = s.statusid
```

```
    WHERE s.Statusid = 4;
```

```
END $$
```

```
LANGUAGE plpgsql VOLATILE
```

## Stored Procedures

### ***Stored Procedures:***

This stored procedure is used when a Person is inserted into the database with all of their corresponding information.

#### **Function Insert\_People:**

```
CREATE OR REPLACE FUNCTION Insert_People("CWID" integer, "First_Name" varchar, "Last_Name"
varchar, "Phone_Number" varchar, "Email" varchar, "PGroup" varchar)
```

```
RETURNS integer AS
```

```
$$BEGIN
```

```
    INSERT INTO People(CWID, First_Name, Last_Name, Phone_Number, Email, PGroup)
```

```
    VALUES ($1, $2, $3, $4, $5, $6)
```

```
    RETURNING CWID;
```

```
END $$
```

```
LANGUAGE plpgsql VOLATILE
```

## Triggers

### ***Triggers:***

This trigger will execute when the `devices_in_progress` function is called to update a possible view of the current devices that are being repaired in the office.

### **Trigger Statement:**

```
CREATE TRIGGER Processes
```

```
AFTER INSERT OR UPDATE
```

```
    ON Device_Status
```

```
FOR EACH ROW
```

```
EXECUTE PROCEDURE devices_in_progress(Int);
```

## Security

### ***Employees Role:***

An Employee at ResNet can insert data into the database; update the data for example, the status of a device in the device\_status tables or lookup statistics of a person who has brought their device to be serviced. An Employee privilege to delete tables in the database has been revoked.

```
REVOKE ALL PRIVILEGES ON People FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Employees FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Employee_Payments FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Customers FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Customers_Request FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Services FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Customers_Process FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Customers_Status FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Status FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Devices FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Device_Types FROM Employee_Privileges;
```

```
REVOKE ALL PRIVILEGES ON Types FROM Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON People TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Employees TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Customers TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Customers_Request TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Services TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Customers_Process TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Customers_Status TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Status TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Devices TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Device_Types TO Employee_Privileges;
```

```
GRANT INSERT, UPDATE, SELECT ON Types TO Employee_Privileges;
```



## Known Problems/Future Enhancements

### ***Final Notes:***

#### **Known Problems:**

- More security enhancements should be implemented on the database to protect customer's information.
- Triggers and stored procedures are minimal in this database and require more research to be effective.
- Since Marist College has different roles in regards to employees, branch tables off of the Employees table should be created to greater specify the difference between full time and part time employees.
  - A security revoke feature should also be implemented onto a possible part-time employees table to greater secure customer's information instead of the current process where all employees (part-time and full) have access to most functionalities of the database.

#### **Future Enhancements:**

- Add support for customers who want to mail in their computer over the summer. This will require implementing more attributes about the user.
- Add a Solutions table section which would connect to the devices and services tables to provide the user with a list of possible solutions when reoccurring problems happen.
- Add more views to calculate statistics for a particular year or month.
- Add more security enhancements to determine who has access to certain tables within the database.
- Add more stored procedures and triggers to the database to increase data integrity and consistency, currently there are only simple functions in place, and it would be optimal to enhance the quality and quantity of these procedures.