Total Mortgage Services

IT Inventory Database

By: Chris Della Donna

Table of Contents

Executive Summary

This database was designed to replace the current spread sheet that currently keeps track of all of the computers and electronics at Total Mortgage Services. It can

effectively manage a lot of the information that they need to, without demanding too much sensitive information. Since they already have all of their computers and other electronics marked with Asset Tags, it was not a difficult decision to make it the primary keys of the assets. Using that as a starting point, the database was mostly built to see who deployed each computer to try to catch mistakes, and it's a far most effective at referencing which user has what equipment and whether that equipment is in or out of the office. Assets are easily able to be added into the system, as well as users, and other devices. With the views and reports that are included, it allows users to access the information that they commonly need for various reasons. Overall, this database is a huge upgrade from the current means of storing data and we are considering putting it to use when I work over break.

ER Diagram

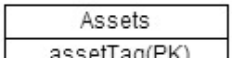

Table Create Statements

Devices Table:

Unique fields include deviceType and deviceDescription. Its purpose it to help categorize the Assets into different types of devices because the company already labels everything with Asset Tags and it is a lot simpler to keep them together.

Create Statement:

```
DROP TABLE IF EXISTS Devices;

CREATE TABLE Devices (

  deviceType              varchar(20) NOT NULL,

  deviceDescription    varchar(100) NOT NULL,

PRIMARY KEY          (deviceType)

);
```

Functional Dependencies:

Column Name	Depends On:
deviceDescription	**deviceType**

Test Data:

	devicetype character varying(20)	devicedescription character varying(100)
1	Desktop	Anything that is considered a desktop computer
2	Laptop	Anything that is considered a laptop
3	Personal Printer	Small Printer on a users desk
4	Copier	Large enterprise copiers
5	Network Switch	Connects multiple devices together via ethernet

Assets Table:

Using the Asset Tags seemed like a pretty obvious thing to start with, especially given that the same information is logged for all of the electronics anyway (make, model, serial number, ETC). This table also offers information such as the date it was purchased and where we bought the item for reference (accounting usually handles the real specific details with purchases so I chose not to include that in the database). This gives us something to look back on if something goes wrong with the product or if we were happy with the experience. All relevant information for all of the assets is stored in this table.

Create Statements:

```
DROP TABLE IF EXISTS Assets;

CREATE TABLE Assets (

  assetTag              varchar(8) NOT NULL,

  deviceType            varchar(20) NOT NULL,

  make                  varchar(20) NOT NULL,

  model                 varchar(20) NOT NULL,

  serialNo              varchar(20) NOT NULL,

  datePurchased         DATE NOT NULL,

  vendor                varchar(20) NOT NULL,

PRIMARY KEY             (assetTag),

FOREIGN KEY             (deviceType) REFERENCES Devices (deviceType)

);
```

Functional Dependencies:

Column Name:	Depends On:
deviceType	**assetTag**
make	**assetTag**
model	**assetTag**
serialNo	**assetTag**

datePurchased	**assetTag**
Vendor	**assetTag**
(deviceType)	**Foreign Key**

Test Data:

	assettag character varying(8)	devicetype character varying(20)	make character varying(20)	model character varying(20)	serialno character varying(20)	datepurchased date	vendor character varying(20)
1	TMS0124	Desktop	Dell	Optiplex 360	123123123123	1993-12-20	amazon
2	TMS0145	Laptop	Dell	Latitude 180	423456233423	2007-04-25	newegg
3	TMS0164	Desktop	HP	ELIE 1337	543464534534	2013-11-01	HP
4	TMS0344	Copier	Ricoh	Copymax 720	7923498792384	2012-04-20	amazon
5	TMS0644	Network Switch	Netgear	Switch Pro	31254234	2011-07-10	Wal-Mart
6	TMS0436	Desktop	Dell	Optiplex 360	3098340982334	2010-04-03	Walgreens

<u>Technicians Table:</u>

The technicians table mostly serves to show which technician deployed which computer in case of future issues or if someone is experiencing a similar issue deploying a particular computer. It's mostly to make sure that Technicians continually upgrade users that remain on older machines with older software (Volume licensing is used for software activation which is the reason why it wasn't emphasized in the design).

Create Statements:

```
DROP TABLE IF EXISTS Technicians;

CREATE TABLE Technicians (

 techId          SERIAL NOT NULL,

 techName        varchar(50) NOT NULL,

 techPhone       varchar(10) NOT NULL,

PRIMARY KEY       (techId)

);
```

Functional Dependencies:

Column Name	Depends On:
techName	**techId**
techPhone	**techId**

Test Data:

	techid integer	techname character varying(50)	techphone character varying(10)
1	1	Chris DellaDonna	2038148273
2	2	Coleman Hathaway	2039278472
3	3	Mike Scott	8369375123
4	4	Santa Clause	8475968375
5	5	Dr. Pepper	8472958639

Locations Table:

The purpose of the locations table is to distinguish between whether an Asset is deployed in the main office buildings or a different location. The business contains a lot of employees that have computers in multiple places. Including this table provides added ease when looking for a specific device's location. Other information can be potentially added to this table, but it isn't particularly necessary since all the assets can be tracked with this small amount of information.

Create Statements:

```
DROP TABLE IF EXISTS Locations;

CREATE TABLE Locations (
  locationType          varchar(20) NOT NULL,
  locationDescription   varchar(50) NOT NULL,
PRIMARY KEY             (locationType)
);
```

Functional Dependencies:

Column Name:	Depends on:
locationDescription	**locationType**

Test Data:

	locationtype character varying(20)	locationdescription character varying(50)
1	On-site	See Users Departme
2	User Home	See Users Address
3	Other location	Check noted record

Buildings Table:

Since the company is run basically in a campus like setting, there are assets scattered about the buildings. In addition, sometimes an order must be placed to one of the other buildings. With this table, the user can quickly see all of the addresses of the offices that we support and it can help to quickly narrow down where to find an employee in need.

Create Statements:

```
DROP TABLE IF EXISTS Buildings;

CREATE TABLE Buildings (

  buildingName                varchar(20) NOT NULL,

  buildingAddress              varchar(50) NOT NULL,

PRIMARY KEY                   (buildingName)

);
```

Functional Dependencies:

Column Name:	Depends On:
buildingAddress	**buildingName**

Test Data:

	buildingname character varying(20)	buildingaddress character varying(50)
1	Main Building	326 West Main Street Milford, CT
2	Executive Building	388 West Main Street Milford, CT
3	Law Office	344 Law Street Fairfield, CT

Departments Table:

The departments table lists the various departments found in the company and where they are generally located. It also leads the technician in right direction, granted the user works on site. It's also helpful when seeking a manager or supervisor in that department.

Create Statements:

```
DROP TABLE IF EXISTS Departments;

CREATE TABLE Departments (
  deptName          varchar(20) NOT NULL,
  buildingName      varchar(30) NOT NULL,
  deptFloor         varchar(20) NOT NULL,
PRIMARY KEY         (deptName),
FOREIGN KEY         (buildingName) REFERENCES Buildings (buildingName)
);
```

Functional Dependencies:

Column Name	Depends On:
deptFloor	**deptName**
(buildingName)	**Foreign Key**

Test Data:

	deptname character varying(20)	buildingname character varying(30)	deptfloor character varying(20)
1	IT	Main Building	Basement
2	Processing	Main Building	1
3	Closing	Law Office	1
4	Marketing	Main Building	2
5	Management	Executive Building	1

Users Table:

The main purpose of the users table is to store commonly referenced data such as the name and contact information of the user. It also keeps track of who currently works from the company and who doesn't work there anymore. Most of this information is noted because many users end up working in their own personal homes with our equipment.

Create Statements:

```
DROP TABLE IF EXISTS Users;

CREATE TABLE Users (
  userId            SERIAL NOT NULL,
  userFName         varchar(50) NOT NULL,
  userLogin         varchar(50) NOT NULL,
  emailAddress      varchar(50) NOT NULL,
  homeAddress       varchar(50) NOT NULL,
  deptName          varchar(20) NOT NULL,
  startDate          DATE NOT NULL,
  endDate            DATE,
PRIMARY KEY        (userId),
FOREIGN KEY         (deptName) REFERENCES Departments(deptName));
```

Functional Dependencies:

Column Name:	Depends On:
userFName	**userId**
userLogin	**userId**
emailAddress	**userId**
homeAddress	**userId**
startDate	**userId**
endDate	**userId**
(deptName)	**Foreign Key**

Test Data:

	userid integer	userfname character varying(50)	userlogin character varying(50)	emailaddress character varying(50)	homeaddress character varying(50)	deptname character varying(20)	startdate date	enddate date
1	1	John Scott	jscott	jscott@totalmortga	323 ilivehere lane	Processing	2000-12-	
2	2	Phil Maas	pmaas	pmass@totalmortgag	23 Liveup Dr.	Marketing	2005-10-	2007-11
3	3	Matt Homes	mhomes	mhomes@totalmortga	834 Trumbull drive	Closing	2009-04-	
4	4	Nick Depizza	ndepizza	ndepizza@totalmort	23 Straight Chilli	Management	2005-03-	
5	5	Joe Archer	jarcher	jarcher@totalmortg	21 Boondocks Ct.	IT	2003-05-	
6	6	Rich Laio	rlaio	rlaio@totalmortgag	83 Not Broken Driv	Closing	2005-03-	
7	7	Steve Omeara	someara	someara@totalmortg	11 Playing LoL Ln.	Management	2012-03-	

<u>Deploys Table:</u>

The Deploys table bridges the gap between Asset and the user that is going to. It allows the IT to easily keep track of who has what equipment, and where that equipment should be given there is a loss of employment or service is needed on a particular device. It keeps track of the deploy details, the hardware involved, and who performed that deploy.

Create Statements:

```
DROP TABLE IF EXISTS Deploys;

CREATE TABLE Deploys (
  deployId              SERIAL NOT NULL,
  assetTag              varchar(8) NOT NULL,
  userId                int NOT NULL,
  locationType          varchar(20) NOT NULL,
  deployDate            DATE NOT NULL,
  restockDate           DATE DEFAULT NULL,
  techId                int NOT NULL,
PRIMARY KEY             (deployId),
FOREIGN KEY             (assetTag) REFERENCES Assets(assetTag),
FOREIGN KEY             (userId) REFERENCES Users(userId),
FOREIGN KEY             (locationType) REFERENCES Locations(locationType),
FOREIGN KEY             (techId) REFERENCES Technicians(techId)
);
```

Functional Dependencies:

Column Name	Depends On:
deployDate	**deployId**
restockDate	**deployId**
(assetTag)	**Foreign Key**
(userId)	**Foreign Key**
(locationType)	**Foreign Key**
(techId)	**Foreign Key**

Test Data:

	deployid integer	assettag character varying(8)	userid integer	locationtype character varying(20)	deploydate date	restockdate date	techid integer
1	1	TMS0124	1	On-site	2001-01-0	2002-02-02	2
2	2	TMS0145	2	User Home	2008-08-1	2010-04-30	1
3	3	TMS0124	4	On-site	2004-08-0		5
4	4	TMS0164	3	Other location	2010-03-2		4
5	5	TMS0344	5	On-site	2001-01-0		4
6	6	TMS0145	6	User Home	2013-01-0		2

Views:

Total Asset Inventory:

The first view that I thought would be useful was a total inventory of all of the assets that we have recorded and the quantity of them based on the Make and Model of the asset. In addition, the accounting department always wants to be kept up to date of how much assets we have in order to pay the correct amount of taxes and ETC.

```
CREATE VIEW Current_TotalInventory AS
SELECT      Assets.make, Assets. model, COUNT(*) Total#Units
FROM        Assets
GROUP BY    Assets.make, Assets.model;
```

Total Available In Stock:

Next, I created a view that displays the Assets that are not currently deployed to a user in order to be sure of what the company has before more electronics are bought for no reason. It will also help the IT staff gauge whether we actually need to order more computers or whether we can get by on what we already have.

```
CREATE VIEW Current_InStock AS
```

```
SELECT Assets.Make, Assets.model, count(*) AS numInStock
FROM        Assets
Where       Assets.assetTag in(
SELECT      Assets.assetTag
FROM        Assets
Where       Assets.assetTag not in(
Select      Deploys.assetTag
FROM        Deploys,Assets
Where       Assets.assetTag = Deploys.assetTag
AND         Deploys.restockDate IS NULL))
GROUP BY    Assets.make, Assets.model;
```

Reports:

Technicians who deployed users on a certain date:

```
CREATE VIEW Tech_Jobs AS

SELECT      Technicians.techName, Deploys.deployDate, Users.userFName,
            Assets.assetTag, Assets.make, Assets.model

FROM        Technicians, Deploys, Users, Assets

Where       Technicians.techId = Deploys.techID

AND         Deploys.userId = Users.userId

AND         Deploys.assetTag = Assets.assetTag;
```

My first report shows which Technician deployed a certain computer to a certain user. As said before, it is useful for quality control and can lead to more ease when it comes to troubleshooting if you seek someone who encountered a similar error/problem with a different device.

Users who currently work at the company:

Since all of the users are permanently stored in the Users table, there are seldom ways to only view the active employees. This could be especially useful in finding contact information for an active employee who requires assistance. The same information is necessary when trying to track down a past employee in hopes of getting equipment back.

It would only require changing the report from looking where the Users.endDate is NOT NULL and instead searching for the users who have an endDate.

```
SELECT      Users.userId, Users.department
FROM        Users
WHERE       Users.endDate IS NULL;
```

Stored Procedures:

Print out Department Roster:

This procedure would print out the Full Names, email addresses, and department of Users that work in a particular department of your choice. Its primary use is also for communication purposes in case there is a need to get in touch with a user.

```
CREATE FUNCTION     departmentRoster(Department.deptName)
Returns table       ( Name, emailAddress, department) as $$
SELECT              Users.FName, Users.emailAddress,  Users.department
FROM                Users.deptName
WHERE               Department.deptName = Users.deptName;
```

```
$$language 'sql';

Select * from departmentRoster('Department.deptName')
```

Triggers:

Verifying that the number of Assets does not go below Zero:

I thought this was a nice thing to include because if someone accidently tried to get ahead of schedule with deploys, meanwhile you have a new user waiting on a computer that you've already reserved a Desktop or something in the system. Since everything is labels and accounted for, it would eliminate this kind of confusion if someone happens to make a mistake while using the database and accidently set the inventory incorrectly low (prevents the inputs of bad data).

```
CREATE Function        Check_Inventory()

RETURNS                TRIGGER AS $$

BEGIN

IF                     (Current_InStock.numInStock < 0) THEN

RAISE EXCEPTION        'You do you not have any more of those available';

END IF;

END $$Language plpgsql;
```

```
Create Trigger StockLow

After insert or update

on Deploys

For Each Row

Execute Procedure Check_Inventory():
```

Verifies that terminated users have retuned their equipment before an official endDate can be set in the database:

This will help account for equipment deployed to users, especially ones that do not work on-site. If the user types an endDate before entering a restockDate, an error will be thrown and they will be prompted for the restockDate before that user can be removed from the system.

```
CREATE FUNCTION NotifyUser()

Returns          trigger as $$

BEGIN

IF                ( Deploys.restockDate IS NULL AND Users.endDate IS NOT NULL) THEN

                  RAISE EXCEPTION 'Please retrieve the equipment from this user and add a
        restock date for this deploy';
```

```
        END IF;

        END

$$LANGUAGE plpgsql;


Create Trigger Equipment_Alert

        After insert or update

        On Users

        For Each Row

Execute Procedure NotifyUser():
```

Security:

I decided, since there is a relatively small group of us working in the IT department, it would probably be best to just create 2 levels of users considering there is a network administrator and everyone else is on the same page. I chose to give Users the permission to edit and add files, but only the administrator can delete them. I figure that the normal users (Technicians) will be performing the majority of the data entry and the modifications to files. Administrators, obviously, have full control over the database.

```
CREATE USER TMSAdmin WITH PASSWORD '007';

Revoke all on Devices          from TMSAdmin;
```

```
Revoke all on Assets          from TMSAdmin;

Revoke all on Technicians     from TMSAdmin;

Revoke all on Locations       from TMSAdmin;

Revoke all on Buildings       from TMSAdmin;

Revoke all on Departments  from TMSAdmin;

Revoke all on Users           from TMSAdmin;

Revoke all on Deploys         from TMSAdmin;


Grant insert, update, delete, select on Devices          to TMSAdmin;

Grant insert, update, delete, select on Assets           to TMSAdmin;

Grant insert, update, delete, select on Technicians      to TMSAdmin;

Grant insert, update, delete, select on Locations        to TMSAdmin;

Grant insert, update, delete, select on Buildings         to TMSAdmin;

Grant insert, update, delete, select on Departments      to TMSAdmin;

Grant insert, update, delete, select on Users            to TMSAdmin;

Grant insert, update, delete, select on Deploys          to TMSAdmin;


CREATE USER TMSHelpDesk WITH PASSWORD 'helpme';
```

```
Revoke all on Devices                 from TMSHelpDesk;

Revoke all on Assets                  from TMSHelpDesk;

Revoke all on Technicians             from TMSHelpDesk;

Revoke all on Locations               from TMSHelpDesk;

Revoke all on Buildings               from TMSHelpDesk;

Revoke all on Departments             from TMSHelpDesk;

Revoke all on Users                   from TMSHelpDesk;

Revoke all on Deploys                 from TMSHelpDesk;


Grant insert, update, select on Devices       to TMSAdmin;

Grant insert, update, select on Assets        to TMSAdmin;

Grant insert, update, select on Technicians   to TMSAdmin;

Grant insert, update, select on Locations     to TMSAdmin;

Grant insert, update, select on Buildings     to TMSAdmin;

Grant insert, update, select on Departments   to TMSAdmin;

Grant insert, update, select on Users         to TMSAdmin;

Grant insert, update, select on Deploys       to TMSAdmin;
```

Other Things to Consider:

Implementation:

Implementation should be simple since the database can be run on the quality of computers at this business and there is not much functionality that is overly complicated. I definitely think that it would be a huge improvement over what they currently have in place.

Known Problems:

I don't think there's as many know problems as there are possible footholds that could come later in the future. For example, more triggers and stored procedures could be added to prevent bad input and common mistakes. However, in the environment that I wrote this for, I think I did a great job of noting the information that they definitely want kept for long term reference. Also, due to the type of information that is being stored in the database, I don't think that it will be much of a target.

Future Enhancements:

Some of the future enhancements that could be added to this system might include more customizable views for more specific reports. More information could be kept based on changing company needs and interests. If they decide they would like to keep track of more IT Supplies, the tables could be easily manipulated to account for it. The future functionality, though, would ultimately be up to the company and it would more than likely be refinement and customization rather than major changes in functionality.