**AAA Roadside Assistance database report**

**by**

**Valentin Antunez**

**Abstract**

The purpose for this database is to provide battery information that will help search for vehicle batteries, past invoices of battery sells and information of every technician that works for AAA roadside assistance. The database will have many functions such as viewing past battery sells, the member’s name and vehicle, battery stock, battery details and vehicle details. The first phase of the project will be designing the tables for the database using Microsoft SQL server. A text menu interface will be made using Perl. Then functional testing will be done and is carried out by feeding the input and validates the output from the application. The final product of the database carries out the requirements.

**Introduction**

AAA roadside assistance provides multiple forms of aid for an AAA member such as changing a flat tire, providing a boost to their vehicle, supplying gasoline, and giving a full battery service. AAA prides itself in delivering fast and efficient customer service. The drivers are trained to fully provide sympathetic, understanding and courteous service for each AAA member. One option AAA provides for any AAA member is testing their vehicle’s battery and selling and installing new batteries. The AAA technicians use a battery analyzer, the ED-18, to give an accurate status of the battery.

Not only are the drivers supplied with equipment and tools to provide battery service, but they also have to know which battery is the correct one for each vehicle. So they are given the *AAA Battery Application Guide* that presents accurate battery information for the member's vehicle.The main issue with the application guide book is that it is clumsy to look through and impossible to read if a driver is given a call while on the road. If the driver has to stop and read through the book it prolongs the wait for the AAA member which will lead to frustration. So implementing a database that can hold every vehicle’s battery information will give the technician a better sense of control and it will translate to more efficient service for the AAA member.

Having that easy access to a vehicles battery will make AAA look more professional and a driver will not be frustrated in finding a battery. It will cut down on the cost of producing books, it can be easily updated with the latest battery information and the driver can quickly find out if the battery is in stock to prevent frustration for the member. The database will also contain information about every battery sell and who the member was that bought it and who the technician was that installed it. This database will provide a reliant and easy access to all the technician’s needs for the perfect battery sell.

**Product Functions**

* The database will provide the correct AAA battery for any vehicle
* It will provide information pertaining a battery (e.g price, terminal)
* It will give each user a personal view of the database.
* The technician will be limited to only their information and will be able to
  + View past battery sells
  + View only the member’s name and vehicle
  + View battery stock
  + Search for a battery
  + View battery details
  + View vehicle details
* The manager will be able to
  + Update technician’s name, telephone and driver id
  + View battery stock
  + View every invoice
  + View members name, address, vehicle, battery purchase
  + Update truck id, and truck type
  + View battery details
  + View vehicle details
  + Search for a battery

**User Requirements**

Each user will require to login into the database. The technician will use their call sign and a password to log in, and will be given only information about the AAA members they have serviced, the catalog of vehicles, their battery quantity, and the invoice numbers for their battery sell. The technician will be able to search through the database using the make, model, year, engine size and options of the vehicle. If everything is correct the user will be shown the battery, any alternative battery, the cold cranking amps, the price, if the vehicle is labor intensive and if the driver has the battery.At any time the technician can search for information about any battery. It will always return the type, cold cranking amps, price and terminal type.

The manager will also have to log in and will be given access to all the technician's name, telephone and truck id. Also the manager will be able to see the entire inventory of batteries and will be able to check which technician has which battery. The manager will be given options to search for batteries as well. They can also search for any invoice and will be shown the member information and the technician that installed the battery.The manager will be able to remove or add any new technicians.

They will be able to see which technician drives which vehicle.

**System Requirements**

The database must allow to add new invoices. The database should be able to handle multiple users at a time. The database will hold the truck id and if the vehicle is one of three different types. Wheel lift tow truck, light service car, or a flatbed tow truck. It will contain the invoice which holds the member’s name, address, membership number, vehicle information, purchase total and battery type. It will store the price, CCA, quantity for every type of battery in stock. It will store the quantity of each battery and update the quantity as they are sold.The database will be able to add new vehicles from future years and update the prices and specification of the batteries.

**Interface Requirements**

This database will be hold many different forms of information. The requirement for the database vary because of this. The requirements for this database are as followed.

Since the main reason to have this database is for a reliable and easy way to search the enormous list of vehicles. So an text interface will be implemented using the PERL system with Microsoft SQL Server 2014. The interface will be a menu system and the user will be able to find information by following the commands and entering the appropriate information. It will show a list of different options for the user.

**Product description**

The database will store all the cars from the AAA application guide and will provide the battery that corresponds with each vehicle. The database contains 10 tables.

The battery table holds the quantity for each battery, the battery type, the price, the manufacturer, the cold cranking amps, and the type of terminal. The battery table provides every information that the technician will need to know. The table can be updated with any new information. For example, if AAA adds a new type of battery to the catalog then it will be added to the table and if there is a price change it will be reflected in the table.

The technician table contains an unique driver id, an unique truck id, the technician’s telephone and the name for the towing company that employs them. Each technician is given a call sign from the phonetic alphabet. The call sign will work as their unique driver id. The technician table has three foreign keys one from the driver table, one from the truck table and last one from the towing company table.

The driver table holds the name for each technician and an unique driver id which is reference by the technician table. This table is updatable and any technician's name can be changed or deleted.

The truck table holds the unique truck id and a truck type which are wheel lift (W/L), light service (L/S) and flatbed (F/B). The truck id is reference in the technician table.

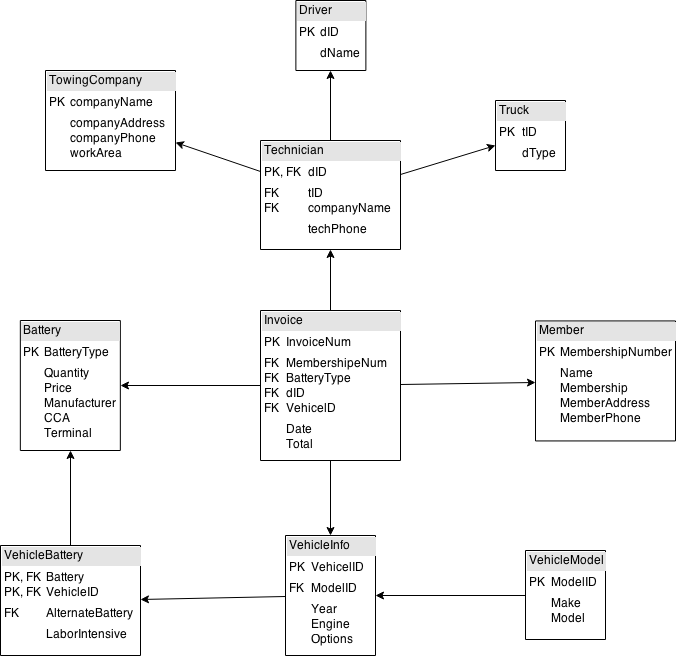
The towing company table will hold the name for the towing companies, their telephone, address and the work area number.

The member table will hold the name, address, membership number, member type and telephone for each member that has purchased an AAA battery for their vehicle. Since there are over a thousand vehicles that can be searched for in the database the vehicle’s information is split into three separate tables.

The vehicle battery table contains an unique vehicle id, the battery that corresponds to the vehicle id, an alternate battery if it is needed, and labor intensive which will indicate if the vehicle with be charged extra for labor. Next is the vehicle information table which contains the year, engine, options, an unique model id and a separate unique vehicle id. The last table is called vehicle model and it holds the make and model of every vehicle and assigns an unique model id which is referenced in the vehicle information table.

The invoice table contains the date, membership number, battery type, driver id, vehicle id, total price, and an unique invoice number. It holds all the information from a battery sell. When the member purchases a battery the technician fills out a physical paper invoice. This invoice will then be added to the table and will be able to show which member has bought which battery and which technician has installed it. It will also contain the member's vehicle information.

**Database Table Structure**



**Table Design for SQL Server**

CREATE TABLE towingCompany(

companyName VARCHAR(100) NOT NULL Primary Key,

companyAddress NVARCHAR(100) NOT NULL,

companyPhone NUMERIC(10,0) CHECK (companyPhone >= 1000000000 AND companyPhone <= 9999999999),

workArea smallint CHECK (workArea >= 10000 AND workArea <= 99999),

);

CREATE TABLE Driver(

dID NVARCHAR(20) NOT NULL Primary key,

dNAME NVARCHAR(MAX) NOT NULL,

);

CREATE TABLE Truck(

tID NVARCHAR(20) NOT NULL Primary Key,

dType NVARCHAR(3) NOT NULL CHECK( dType ='L/S'OR dType ='F/B' OR dType ='W/L'),

);

CREATE TABLE technician(

dID NVARCHAR(20) NOT NULL Primary Key FOREIGN KEY REFERENCES Driver(dID),

tID NVARCHAR(20) NOT NULL unique FOREIGN KEY REFERENCES Truck(tID),

techPhone NUMERIC(10,0) (techPhone >= 1000000000 AND techPhone <= 9999999999),

companyName VARCHAR(100) NOT NULL FOREIGN Key REFERENCES towingCompany(companyName),

);

CREATE TABLE Member(

MembershipNumber BIGINT NOT NULL Primary Key,

Name NVARCHAR(MAX) NOT NULL,

Membership NVARCHAR(10) NOT NULL CHECK(Membership = 'Basic' OR Membership = 'Plus'OR Membership = 'Premier'),

MemberAddress NVARCHAR(MAX) NOT NULL,

MemberPhone NUMERIC(10,0) CHECK(MemberPhone >= 1000000000 AND MemberPhone <= 9999999999),

);

CREATE TABLE Battery(

BatteryType NVARCHAR(10) NOT NULL Primary Key,

Quantity smallint,

Price smallint NOT NULL CHECK( Price = 99 OR Price = 111 OR Price = 148),

Manufacturer NVARCHAR(MAX) NOT NULL,

CCA NVARCHAR(4) NOT NULL,

Terminal NVARCHAR(4) NOT NULL CHECK(Terminal = 'Top' OR Terminal = 'Side' OR Terminal = 'Dual'),

);

CREATE TABLE VehicleModel(

Make NVARCHAR(100) NOT NULL ,

Model NVARCHAR(100) NOT NULL,

ModelID int Identity(1,1) Primary Key,

);

CREATE TABLE VehicleInfo(

ModelID int NOT NULL Foreign Key REFERENCES VehicleModel(ModelID),

Year int NOT NULL,

Engine NVARCHAR(20) NOT NULL,

Options VARCHAR(50),

VehicleID int Identity(1,1) Primary Key,

);

CREATE TABLE VehicleBattery(

VehicleID int NOT NULL Foreign Key REFERENCES VehicleInfo(VehicleID),

Battery NVARCHAR(10) Primary Key Foreign Key REFERENCES Battery(BatteryType),

LaborIntensive BIT NOT NULL,

);

CREATE Table Invoice(

InvoiceNum NVARCHAR(10) Primary Key NOT NULL CHECK(InvoiceNum >= 0 AND InvoiceNum <= 9999999999),

dID NVARCHAR(20) NOT NULL Foreign Key REFERENCES Driver(dID),

Date DATE NOT NULL,

MembershipNumber BIGINT NOT NULL Foreign Key REFERENCES Member(MembershipNumber),

Total smallint NOT NULL,

BatteryType NVARCHAR(10) NOT NULL Foreign Key REFERENCES Battery(BatteryType),

VehicleID int NOT NULL Foreign Key REFERENCES VehicleInfo(VehicleID),

);

The following are a sample of the perl menu design. Using dbi module to communicate to sql server.

sub FindBattery(){

my $sth = $dbh->prepare('SELECT Battery,AlternateBattery AS Alternate, Engine, Options

From BatteryVehicle

WHERE(SELECT ModelID

FROM VehicleModel

Where Model = ? and Make = ?) = ModelID and Year = ? ')

or die "Couldn't prepare statement: " . $dbh->errstr;

print "Enter Make > ";

my $Make = <>;

my @data;

chomp $Make;

$sth->bind\_param( 2,$Make);

print "Enter Model > ";

my $Model = <>;

chomp $Model;

$sth->bind\_param( 1,$Model);

print "Enter year > ";

my $Year = <>;

chomp $Year;

$sth->bind\_param( 3,$Year);

$sth->execute(); # Execute the query

# print "\tBattery :\tAlternate :\tEngine :\tOptions\n";

# Read the matching records and print them out

while (@data = $sth->fetchrow\_array()) {

print "\t@data[0] :\t@data[1] :\t@data[2] :\t@data[3]\n";

}

if ($sth->rows == 0) {

print "No match.\n\n";

}

$sth->finish;

print "\n";

}

sub FindInvoice()

{

my $sth = $dbh->prepare('SELECT Date,InvoiceNum,BatteryType,Total,Year,Make,Model,Engine,Name

From DriversInvoice

WHERE dName = ?')

or die "Couldn't prepare statement: " . $dbh->errstr;

print "Enter Name> ";

my $Name = <>;

my @data;

chomp $Name;

$sth->execute($Name);

while (@data = $sth->fetchrow\_array()) {

print "\t@data\n";

}

if ($sth->rows == 0) {

print "No match.\n\n";

}

$sth->finish;

print "\n";

}

sub BatteryInfo()

{

my $sth = $dbh->prepare('SELECT \*

From Battery

WHERE BatteryType = ?')

or die "Couldn't prepare statement: " . $dbh->errstr;

print "Enter Battery > ";

my $Battery = <>;

my @data;

chomp $Battery;

$sth->execute($Battery);

while (@data = $sth->fetchrow\_array())

{

print "\t@data\n";

}

if ($sth->rows == 0) {

print "No match.\n\n";

}

$sth->finish;

print "\n";

}

sub BatteryStock()

{

my $input = &rl(batteryChoice());

&SpecificBattery() if ($input =~ /a/i);

&AllBattery() if ($input =~ /b/i);

&Update() if ($input =~ /c/i);

}

sub AllBattery()

{

my $sth = $dbh->prepare('SELECT Battery,Quantity

From BatteryStock

WHERE dID = ? ')

or die "Couldn't prepare statement: " . $dbh->errstr;

print "Enter Driver ID > ";

my $dID = <>;

my @data;

chomp $dID;

$sth->execute($dID);

my $Total = 0;

while (@data = $sth->fetchrow\_array())

{

if ($data[1] == 0)

{}

if ($data[1] != 0)

{

print "You have @data[1] @data[0] batteries.\n";

}

$Total += $data[1];

}

print "Total = $Total.\n";

if ($sth->rows == 0) {

print "No match.\n";

}

$sth->finish;

print "\n";

}

sub SpecificBattery()

{

my $sth = $dbh->prepare('SELECT Battery,Quantity

From BatteryStock

WHERE dID = ? and Battery = ?')

or die "Couldn't prepare statement: " . $dbh->errstr;

print "Enter Driver ID > ";

my $dID = <>;

my @data;

chomp $dID;

$sth->bind\_param( 1,$dID);

print "Enter Battery > ";

my $battery = <>;

chomp $battery;

$sth->bind\_param( 2,$battery);

$sth->execute();

while (@data = $sth->fetchrow\_array())

{

print "\tYou have @data[1] @data[0] batteries\n";

}

if ($sth->rows == 0) {

print "No match.\n\n";

}

$sth->finish;

print "\n";

}

sub Update()

{

my $sth = $dbh->prepare('UPDATE BatteryStock

SET Quantity = ?

WHERE Battery = ? and dID = ?')

or die "Couldn't prepare statement: " . $dbh->errstr;

print "Enter Driver ID > ";

my $dID= <>;

chomp $dID;

$sth->bind\_param( 3,$dID);

print "Enter Battery > ";

my $battery = <>;

my @data;

chomp $battery;

$sth->bind\_param( 2,$battery);

print "Enter Amount > ";

my $amount= <>;

chomp $amount;

$sth->bind\_param( 1,$amount);

$sth->execute()

or die "Couldn't prepare statement: " . $dbh->errstr;

if ($sth->err)

{

die "ERROR!" . $sth->err . " error msg: " . $sth->errstr . "\n";

}

else {print "Done!\n"}

$sth->finish;

}

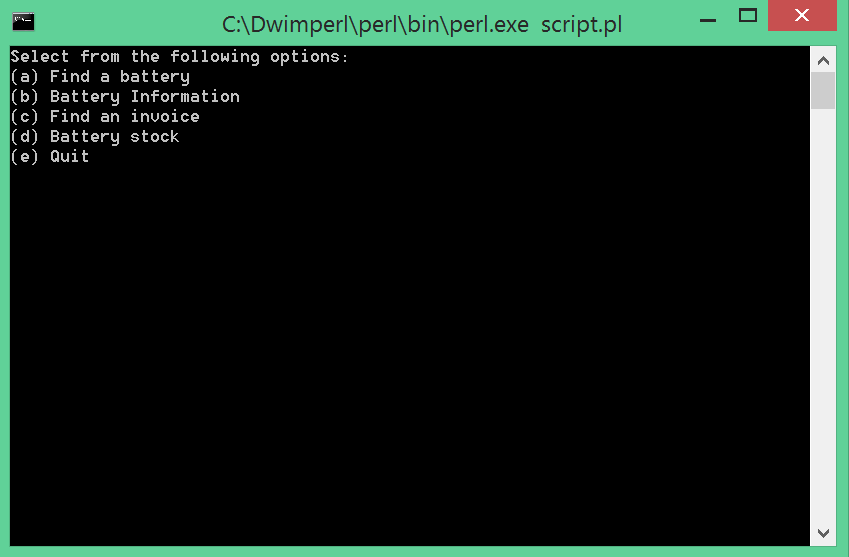
**Test Plan**

The objective of the test is to verify that the functionality of the Perl interface works according to the specifications.

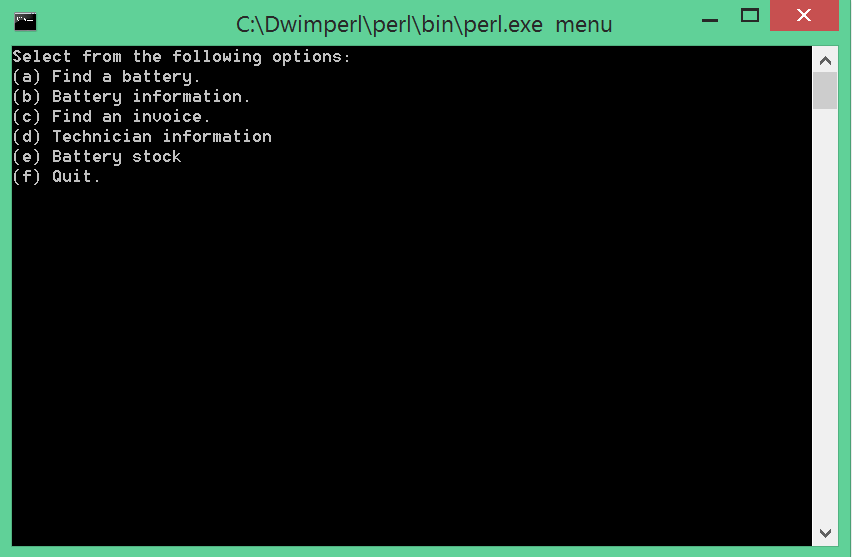
**TESTING STRATEGY**

The overall approach to testing this database is to enter as many queries to find the correct answer. Each function of the database will be tested. Searching for a battery will be tested by entering random vehicle information from the AAA application guide and matching the result to the book. This approach will be repeated for all search functions. The update battery stock function will be tested by entering random details some will be false and verifying that the database interacts correctly.Each function will be run at least once and any errors will be written down. The user requirements will be tested. Functional testing will be performed to check the functions of database. The functional testing is carried out by feeding the input and validates the output from the application.

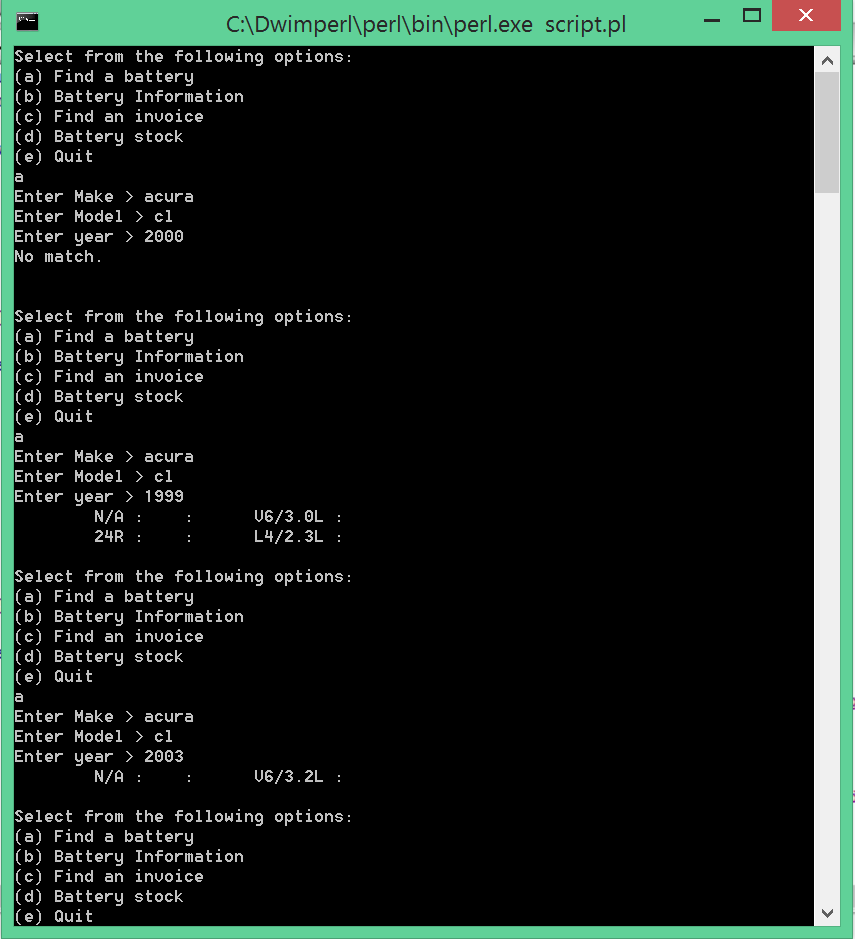
**Results**



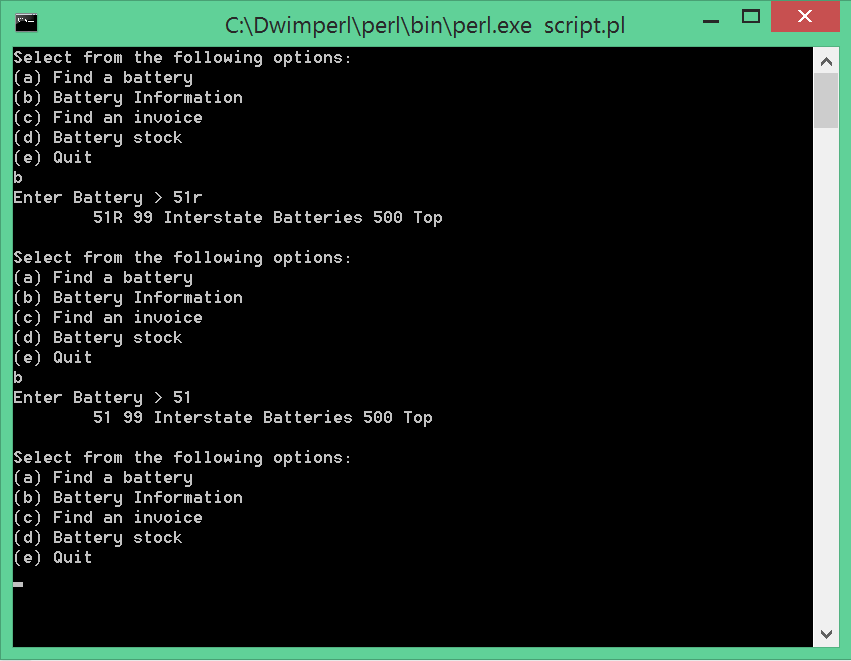
Technician’s menu



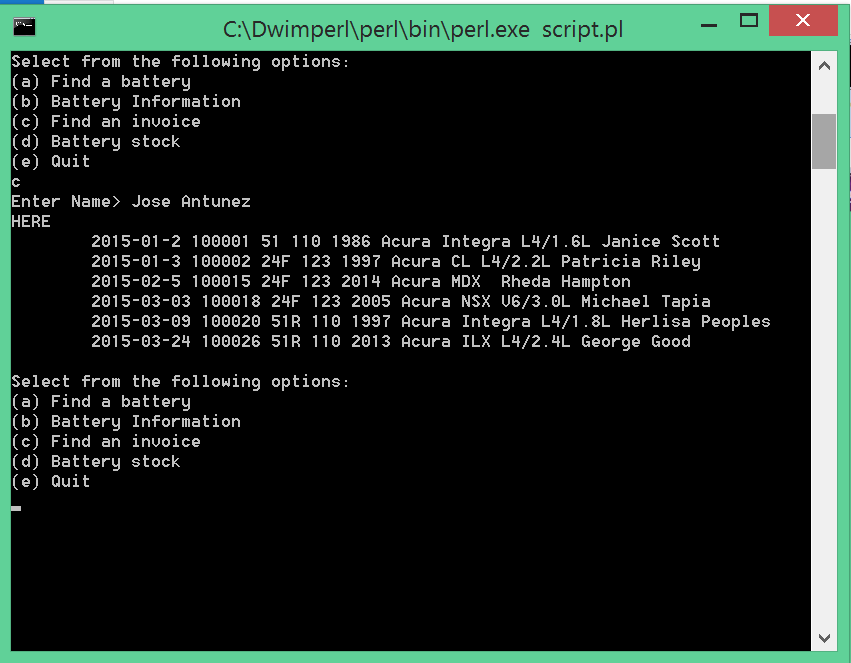
Manager’s Menu



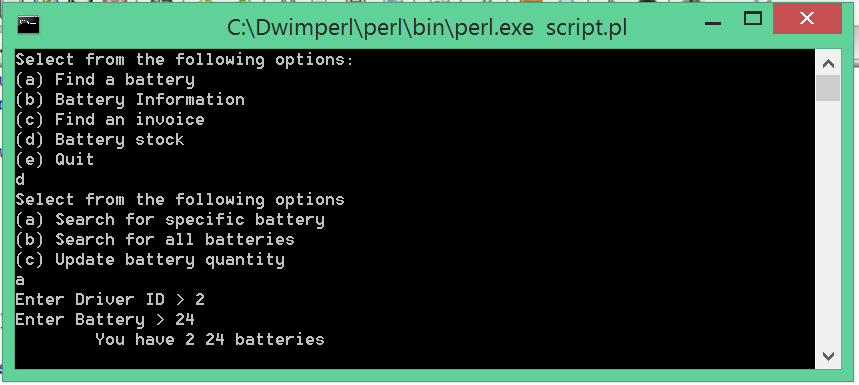
Finding a battery



Finding Battery Information



Finding invoice



Finding Battery Stock

**Conclusion**

The database shows all the batteries that correspond with the vehicle entered. It displays the invoice that only correspond with certain technicians. It lets the user update individual battery quantity and displays the battery stock for either individual batteries or every battery. It only displays results from specific drivers.When testing the database and the menu the results are very promising.

**References**

Bunce, Tim. Programming the Perl DBI. N.p.: O'Reilly Media, n.d. Print.

Dominus, Mark-Jason. "A Short Guide to DBI." Perl.com. N.p., 22 Oct. 1999. Web. 1 Apr. 2015.

"SQL Tutorial." SQL Tutorial. Refsnes Data, n.d. Web. 3 Mar. 2015.