Hunter Kendall

Mac Tools Database

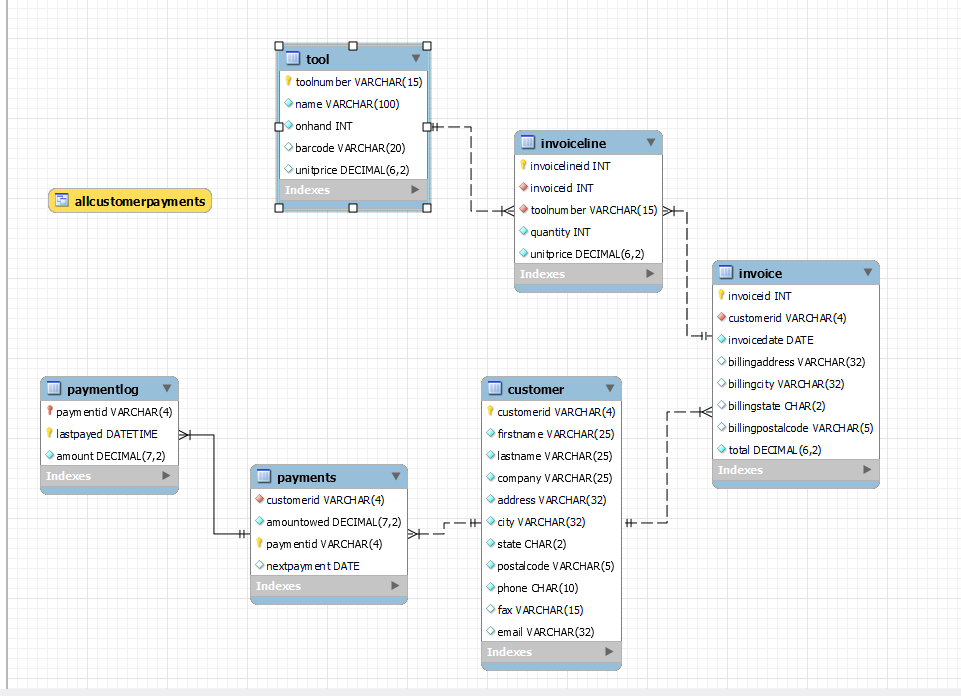
Overview of the Project:

When starting this project I was not sure what I wanted to do. After Thinking for a while I remembered that my dad has his own Mac Tools distributor business and there was a database to go with it. Essentially he drives to various shops companies in a large truck, that is a mobile “store,” and he sells tools out of it. After helping my dad with inventory which takes a couple of days since it is all done by hand, (reading numbers and putting them into an application), it dawned on me that i could make my own version of the database that uses the barcodes on the products. This would decrease the amount of time it takes to take inventory. The database also needs a way to add the new tools that are released every year. This will keep what tools   
 While talking with my dad sometimes he brings up customers that he is a little frustrated with because they do not pay very much while paying for tools they buy. So to add a little more substance to the database I created a way to see how much a customer owes you and tracks when they paid you and the amount.

The database will also be able to return an xml version of all the customers at a single company. This will be used when arriving at a company we can look up who we expect to do business with and see who owes money and when their next payment is due.

Lastly the UI needs to be pretty simple for viewing certain data so there will need to be some prebuilt queries. This is because the user is not expected to know SQL language to write queries.

Database Schema:



Functional Dependencies and Keys:

For relation Tool:

Primary key: toolnumber

List of Functional Dependencies:

* toolnumber ➝ name, onhand, barcode, unitprice

toolnumber is a unique id given to each tool in the database

* name ➝ toolnumber, onhand, barcode, unitprice

Name is also a unique attribute but since it is so long its hard to search with.

For relation InvoiceLine:

Primary Key: invoicelineid

Foreign Keys: toolnumber, invoiceid

List of Functional Dependencies:

* Invoicelineid ➝ invoiceid, toolnumber, quantity, unitprice

Invoicelineid uniquely identifies each item bought in each invoice.

For relation Invoice:

Primary Key: invoiceid

Foreign Key: customerid

List of Functional Dependencies:

* invoiceid ➝ customerid, invoicedate, billingaddress, billingcity, billingstate, billingpostalcode, total

invoiceid uniquely identifies who bought what when and who to send this information too.

* billingaddress, billingcity, billingstate ➝ billingpostalcode

billingaddress, billingcity and billingstate all reside inside a single billingpostalcode.

For relation Customer:

Primary Key: customerid

List of Functional Dependencies:

* customerid ➝ firstname, lastname, company, address, city, state, postalcode, phone, fax, email

customerid is a unique id given to each customer to uniquely identify all the data stored for them

We can keep company in because we do not charge the company just the individual customer. Company is used to show who we expect at a shop.

* address, city, state ➝ postalcode

Address, city and state all reside inside a single postalcode.

For relation Payments:

Primary Key: paymentid

Foreign Key: customerid

List of Functional Dependencies:

* customerid ➝ amountowed, paymentid, nextpayment

customerid is a unique identifier given in the customer relation and is used here to identify how much is owed, assignment of their paymentid and tell us when that specific customer's next payment is due.

* paymentid ➝ amountowed,customerid, nextpayment

Paymentid is also a unique identifier which tells the who made the payment(customerid), how much is left (amountowed), and when the next payment is due.

For relation Paymentlog:

Primary Keys: paymentid, lastpayed

Foreign Key: paymentid

List of Functional Dependencies:

* paymentid, lastpayed ➝ amount

paymentid tells us who payed, lastpayed tells us when paymentid payed which will always be unique since a specific customer cannot pay twice at the same time.

Normal Forms:

Relation Tool: is in bcnf

* Both toolnumber and name are both super keys and since those are the only functional dependencies thus this relation is in bcnf.

Relation InvoiceLine: is in bcnf

* invoicelineid is a super key and since that is the only functional dependency thus the relation is in bcnf

Relation Invoice: is in 2nf

* Invoiceid is our super key and candidate key of the relation. Since billingaddress, billingcity, billingstate, and billingpostalcode are all attributes of invoiceid they show transitivity. Thus the relation is not in 3nf.
* Since billingaddress, billingcity, billingstate is not a subset of candidate key invoiceid thus the relation is 2nf.

Relation Customer: is in 2nf

* customerid is a super key and candidate key of the relation. Since address, city, state, and postalcode are all attributes of invoiceid they show transitivity. Thus the relation is not in 3nf.
* Since address, city, state is not a subset of candidate key customerid thus the relation is in 2nf.

Relation Payments: is in bcnf

* Both customerid and paymentid are both super keys and since those are the only functional dependencies thus this relation is in bcnf.

Relation Paymentlog: is in bcnf

* paymentid, lastpayed is a super key and since that is the only functional dependency thus the relation is in bcnf.

Implementation Issues:

There were a couple of implementation issues/complications.

Purchase function:

The purchase function had some issues with writing the invoices in the invoice line. The issue arose when I was almost done with the project. I noticed that invoice total was zero because I was inserting that into the relation before the total was computed. Easy fix just iterate through the cart of items and compute the total before writing the invoice. This caused the invoiceline relation to not have any data to be input. The reason or this is because when I make the cart I would store it as a zip. I do not use zips all that often so I was unaware that when a zip is used it clears the zip. This meant that when I tried to iterate through the cart again to insert the invoiceline the cart would be empty, thus nothing was being inserted. To fix this I converted the zip to a list.

DATETIME in Paymentlog:

When using the payment function multiple times in a row there is a bug with the time in date time. I'm not sure where it is but the time is always the same after the first purchase is made during the running instance The datetime is received from the python package datetime. What is called to get the DATETIME element is d\_time = dt.now() (dt is stands for datetime but since the package is also datetime i renamed it as dt). Why it does not update d\_time every time is strange to me as d\_time is a default function parameter. So if a customer is to make 2 purchases in the same day (running instance) it would cause an error of a non unique primary key

Barcode Scanner implementation:

The barcode was not necessarily an issue, it is one of the main points of my project. I found a project using the python library from the website link below and modified it to fit my needs. The way their project works is that it runs a while loop constantly scanning and returning the barcode. I needed a return of one barcode string. I implemented this by scanning 5 times and checking that each scan was the same. Then returned the barcode as a string. I use a string because not every barcode returns numbers as some have letters in them. Also if the barcode started with a zero it would remove all the zeros before the first number greater than zero.

I also turned this project into a class and put it into its own separate file so it can stay separate from the application functions. This allows it to be able to be used in future applications if needed.

Project source: <https://kalebujordan.com/reading-bar-codes-python/>

Project Evaluation:

Overall I think I had a fairly decent project outcome. With time constraints and other conflicting schedule changes I am happy with what I came up with. I feel I have met my goal that I started at the beginning of this database project. These goals include:

* Taking inventory by using a barcode scanner.
* Logging customers transactions and payments.

Project requirements met:

* 4 - 8 relations (have 6)
  + Primary/Foreign key pairs
* At least 10-15 attributes (Have 36)
* Proper domains for all attributes
* Adequate constraints
* Data separation (customer data is not in payment table and etc.)
* Input/update (Adding tools and taking inventory)
* Can export customer data for a company in xml
* Select (used in most functions and is in data tables option in the UI
* A join is used in the payment function and export function.
* I have 1 meaningful view(proposal for requirement change)
  + Most of the functions under data tables in the UI return either a smaller version of the relation they return instead of having multiple views that aren't super meaningful. For example Function company list is better off as a Select from parent tables since we would need to make a separate view for each company which does not seem realistic.
* Aggregation Operators (in my one view and when looking for last payed.
* Transaction...commit (in most of my functions)

Project requirement not met:

* Repetitive data (possibly)
  + In my payments relation I give each customer id a paymentid. At the time of creating the relations this seemed like a good idea as what if the paymentid’s and customerid’s get out of sync because of a delete. This would not happen if i used a foreign key from Customer.customerid to Payments.paymentid then removed customerid from the Payments relation. So now we have two elements in each tuple that are exactly the same.

Lessons learned:

* I learned Mysql to python connector syntax or multiple ways to write the same code.
* I learned how to properly set up primary and forign keys.
* I learned how to extract data from a database, manipulate it and return it into the database.
* I learned how to use the python library pyzbar to read barcodes.

Next Steps:

* Creating a proper GUI for the user.
* Get feedback from someone who will use said database for what they want to show in the Data Tables.
* Creating a better assignment of customerid. The current one can be messed up if a tuple is deleted.