Database Assignment

### Purpose and End User of my database

| The purpose of my database is to see what toppings,sauce and patties and in what combination is the most popular and will be enjoyed by the most people. My end user are people who order burgers and the people selling it and by using the database I can see what is popular and potentially improve. |
| --- |

### Describe at least 3 implications that are relevant to your database and its use by the end user and why they are important

| Privacy:  Privacy is the customers information such as address,phone number ,password and more. Privacy is important because it is what the customers don’t want others such as hackers and stalkers and third party to see. For example you wouldn't want your phone number to be found and then getting bombarded with hacker numbers everyday. This is why privacy is very important because if many people knows about people’s privacy info getting stolen people might not trust the people keeping the data.  Usability:  Usability is how easy an order can be stored in the data base. It must be easy to use and not take a long time to find so the user can place an order very easily. Usability is very important because customers like it to be easy to make an order and don't like hassle. For example nobody likes a website that takes forever to find something and is all over the place. This is because when people are trying to place an order they would like it to be quick and easy to use so it uses as least amount of their time as possible and also the data base should also be reasonably tidy and sorted so people can see the orders and make charts determining what is popular and what isn’t.  functionality  Functionality is does the ordering database work and does it have bugs. Functionality is is the data base buggy and does it crash or work properly like intended for it to. For example is there a bug where people’s database is deleted by accident or duplicated. This is important for the data base because they things could cause false information and maybe also break the database.  S-state  E-explain  X-example  Y-why |
| --- |

### Database Design- Your Entity Relationship Diagram.

|  |
| --- |

### Database Testing Table: SQL Statements

| **Purpose** | **SQL Statement** | **Result Success?** |
| --- | --- | --- |
| To list all the sauces | SELECT name FROM sauce |  |
| To list all the patties | SELECT name FROM patties |  |
| To list all the toppings | SELECT name FROM topping |  |
| Shows the name and the rating from patties ordered by worst to best | SELECT name, rating FROM patties ORDER BY rating |  |
| Shows the name and the rating from sauces ordered by worst to best | SELECT name, rating FROM sauce ORDER BY rating |  |
| Shows the name and the rating from toppings ordered by worst to best | SELECT name, rating FROM toppings ORDER BY rating |  |
| Shows the name , price, rating of patties and orders them by the best rating for the for the cheapest price | SELECT name,price,rating FROM patties ORDER BY price,rating |  |
| Shows the name , price, rating of toppings and orders them by the best rating for the for the cheapest price | SELECT name,price,rating FROM topping ORDER BY price,rating |  |
| Lets the customer add beef into their order | insert into patties(name, price, rating) values('beef', 4, 9); |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Relevant Implications- Explain how your database addresses the relevant implications that you identified at the start.

| My database adresse the privacy part for relevant implications by keeping the names of the customers stored and not shared. My database’s Usability is great because it is very easy to use as you can order items or change items very easily and quickly. The Functionality of the database is also good as it doesn't have any bugs which could disrupt the functioning of the database. It adresse the usability implication by having easy accessibility to order and edit items, it also addresses the functionality implication by having try and while loops which will make the thing repeat itself if there is a mistake. |
| --- |

### Showcase:

Give evidence of your database and the Python code that interfaces with it. Use screenshots or a short video. Explain how it improved, how it functions, how it was tested etc.

| <https://replit.com/join/wtbnalcxka-mengdemo>  TESTING:    This part of the code tests if each function is working and excludes it from the interface so I could test it faster. I use ''' to make all the main interface comments so it will only run the testing code.i can test the functions one at a time and test each function if there is an error or just to test it quickly without going through all the main code  HOW the Customers and staff member are different.      If you enter 1 please read what happens when you press 1 (CUSTOMER)  If you entered 2 please read what happens when you press 2(STAFF MEMBER)  FUNCTIONS:  CUSTOMER:  Show\_burger\_info()    This is the code for show\_burger\_info()  It only asks for what you wish to view out of patties,toppings and sauce and whether you would like to continue or not. It use a different sql for different options and is very simple but is durable and if you enter a non number value or a number that is not 1,2 or 3 it will say and INCALID ANSWER    Here is what happens when an person enters something stupid like fish or something. It will first trigger the try command from the show\_burger\_info function and go from executing the function to the except command which only happens when an error has occurred inside the function and will then print INVALID ANSWER and then due to the while loop it will ask the question(repeat the function) again. It will keep doing this until you enter 1,2 or a 3.  pattie:      At the start it will ask you what would you like to see out of the pattie,toppings and sauce. Depending on the answer you choose the sql will be different. For example if you typed in “1” the sql would be “SELECT \* FROM patties” and then it will print out the rows for ID,Name,Price and rating after that it will ask you whether you wish to continue viewing or not.  Topping:    It will ask you the same question”what would you like to see” with the same options patties,topping and sauce but if you enter 2 this time you will see the toppings with it’s ID,name,price and rating. The sql when 2 is entered will now be “SELECT \* FROM topping” instead of above where it was “SELECT \* FROM patties” because options\_pts is = to “2” this time so the sql is now “SELECT \* FROM topping” instead of “SELECT \* FROM patties”.  Sauce:      If this time you options\_pts is equal to 3 the ID,name,price and rating for sauce will appear. This time since 3 is entered instead of 2 or 1 the sql will be “SELECT \* FROM sauce” instead of “SELECT \* FROM patties” or “SELECT \* FROM topping” This is because since the sql is now assigned as “SELECT \* FROM sauce so when the cursor,execute(sql) happens since it will print out all the data for sauce instead of patties and topping.  Order\_item()      This is the code for the 1st image(how many burgers), It only include the patties order but the others are the same in code except for the difference in some variable names and the sql difference but apart from that , there is nothing different and they all do the same thing. The first thing it does is ask you how many burgers you want,1-5, it then checks if your answer is between 1-5 and if it isn't it will trigger the else: at the last image and if it is not a number it will trigger the except command see last image for both. Lets say you do enter something between 1-5 it will then set num\_burg=0(“its just a var not important”) , the var num\_burg is only used to change X to 1,2,3,4 or 5 (BURGER-X). After printing which burger it is on it will start the code for patties.      For the checkout it first summarizes all the burgers with their patties,topping and sauce and then asks you to pay for the combined sum of all burgers. How it works is that it has another pretty useless var called num\_burg2(“just another var for printing burger-X (X =1,2,3,4 or 5) But first it uses the var num\_burger which was at the start(asks how many burger you want) and it uses that to determine how many times to loop everything inside the for loop. Then it prints out in the list called burger(patties,topping,sauce) the number (num\_burg2) one so if num\_burge2 = 1 it would print out the 2nd item inside (as python list [0]= first item). It then prints all the prices of the sauce,patties,topping of all the burgers which are all in one big list and by using the sum command I can just add them together in one go and it will then print out what the total is.    If the first IF statement is not met(if num\_burger >0 and num\_burger <=5:) which just checks if the amount of burgers you want is between 1 and 5. If this statement is not met it will then trigger the else: statement which will print(“INVALID CHOICE . And since the whole function is in a while loop it will restart the whole thing and ask you again therefore making it bulletproof. It also will print”something went wrong or INVALID choice \n try again and then restart the questions if the user breaks the code for example enters a non existent pattie or enters a string into a int(input) where they ask for numbers. This makes the code even more bulletproof.  ordering      This ask you how many burgers you want and also num\_burger will be used in later code for determining how many times to do a for loop. The Maximum is 5 as we want it so that everyone can have burgers and someone doesn't just order it all.    Because we choose to order two burgers we have the two burgers(burger 1 and burger 2)  We even get to see the name price and rating for each pattie,topping and sauce we choose.  How the what to order works:      All that code under #patties order is for this few lines. It may look simple but a lot is actually going on when you are ordering what patties/topping/sauce. For this we will look at the code for patties because they are all the same except for the sql,variable names and what list the name gets appended to but those are not very important and the main code is the same for all of them. Firstly it sets the sql to select the names,prices and ratings of all the patties. It then repeats checking if each row’s first value(name) is == to what you typed in item\_to\_find(what pattie would you like) and if you found it it appends that name to a list called burger\_patties which will be used at checkout and it also appends the price to a big list with all prices and that will also be used for checkout. After that found\_pattie(a var) is = 1 and the for loop is finished. After that we have an if statement checking whether found\_pattie is still 0(because if it is it means that in the for loop they couldn't find and item that matched the name of the pattie you entered.) if there is no such name as the one you entered in patties means that the name you entered doesn't exist and it will break out of the while loop. As shown in the image below, NOTE”not beef” does not exist as a pattie(just an example)    Variables    These are the variables/lists used for storing burger1’s patties,topping and sauce as (list\_name[0])  Also known as the first item in the list. If you ordered more than 1 burger 2,3,4 or 5’s patties,topping and sauce will be the(list\_name[1,2,3,4]. There is one list that is different from the others and that is the tot\_burger\_prices. This list gets the prices from the patties,topping,sauces you order for the burgers but unlike the other since the price is combined at the end it is fine to just put them in one big list and then use the sum command to just instantly sum everything together.  STAFF MEMBER:    What first happens is it asks you for the password(fish), this is because you wouldn’t want anyone to beagle to us the add\_item(),delete\_item(),update\_item() and show\_orders(). THis is because if they could some idiots could just delete everything or add a bunch or random menu items that are silly or don’t exist or they could change the price of items to 1 each and that would ruin the business. We also definitely can’t show them show\_orders() because it contains info about the customers and what they ordered and some customers might not want that information leaked and that is why we have a password.      After that we have 6 options to choose from(add,delete,update,view[items] or show customers and their orders and leave) if one of the numbers from 1-6 were not entered it will print INVALID ANSWER  If a string is entered instead of a number the except will activate and print invalid answer and also since everything is is a while loop it will restart.  After this read about what each of the others do read FUNCTIONS:  FUNCTIONS:  add\_item()      This is the main part for the add(before #topping table). All it does is ask you the name price,rating of the new item and then asks you which table it will go into. After that depending on which table you put it in(eg-toppings) the sql is “INSERT INTO topping (name,price,rating) Values(?,?,?)” what this does is then cursor.execute(sql,(item\_name,item\_price<item\_rating)). What this does is using the values that you imputed in at the start it will decide which table to put the new item and what the name,price and rating are. After that it then prints out the new updated table with your new item.  View below,that the one we added(example) is at the bottom(11)  delete\_item()    How the delete\_item() function works is that it first asks you for the name of the item you want to get rid of. Then it asks you in what table you want to get rid of it from.(this is main code) after that if we wanted to delete it from toppings the sql will be set to “DELETE FROM topping WHERE name = ?”  And what that does is whatever the name you wanted to delete it will look for it in the table you specified and if it finds it it will delete that item but if there is no such item(see checking(below)).  After that it will then show you the table with the item you specified from that table and it will no longer be there.    Note: look at the picture from add\_item() compare and you will see that example is now gone.  Checking:    What this does is if the rows affected by the sql is == 0 it will print can’t find that item signifying that the item does not exist or it’s not in this table.    update\_item()    The update\_item() is very similar to the delete\_item() in terms of code with only a few noticeable differences, them being the sql(“UPDATE topping SET price = ?,WHERE name= ?”) . What happens is that it has 2 unknowns which is the name and the new\_price. Which you will enter in at the start(update\_price and Item\_name) Then depending on what table you choose to update it from (this example=topping). Then it checks how many rows were affected if it is = to 0 meaning the name does not exist it will print cant find item then break meaning it will start the function again. But if it does succeed it will print your item has been updated from the menu and the menu where you can check the new updated price with all the other items.  Before:    After:      Show\_orders()    All show\_order() does is shows the ID of the customer along with the pattie,topping,sauce IDs along with it. The sql is just “SELECT \* FROM burger\_info” what this does is shows the customer orders so far. After that all it does is just ask you whether you want to continue viewing or not.    Show\_burger\_info()    This does exactly the same as the show\_burger\_info in Customer: but it is also an option for staff members as it would be inconvenient to view the menu by going to the customer side and being able to do it in staff member: is a lot faster and simpler.    Leave:    If options is 6 you leave the staff member mode and it will ask you the whole thing again.    CHECKING:    If at any point the inputs the user enters a mistake or a string in a int(input) or something that doesn't exist the code, because I have a try and while loop at the start what will happen is if it is an input question and you enter an non existent answer that code will go to the else but if you enter a string in an int(input), and since we have the try command the excpt command will then be activated and print(“INVALID ANSWER please try again.”) |
| --- |

**Teacher Checklists:**

**AS91879- Develop a digital outcome to manage data**

**Credits: 4**

**NZQA:** [https://www.nzqa.govt.nz/nqfdocs/ncea-resource/achievements/2019/as91883.pd](https://www.nzqa.govt.nz/nqfdocs/ncea-resource/achievements/2019/as91883.pdf)f

| Achieved- Develop a digital outcome to manage data | **Evidence** |  |
| --- | --- | --- |
| using appropriate tools and techniques to structure, organise, query and present data for a purpose and end user |  | ✓ |
| applying appropriate data integrity and testing procedures |  | ✓ |
| describing relevant implications. |  | ✓ |
| Merit- Develop an informed digital outcome to manage data |  |  |
| using information from testing procedures to improve the quality and functionality of the outcome |  | ✓ |
| structuring, organising and querying the data logically | But where was the joins? What about seeing the entries in the burger\_info table? | ✓ |
| addressing relevant implications. |  | ✓ |
| Excellence- Develop a refined digital outcome to manage data |  |  |
| iterative improvement throughout the development and testing process |  | - |
| presenting the data effectively for the purpose and to meet end-user requirements. | It’s not user friendly enough for E | - |

**Develop a computer program**

**Credits:** 4 (Internal)

**NZQA:** <http://www.nzqa.govt.nz/nqfdocs/ncea-resource/achievements/2018/as91883.pdf>

| **Achieved**  **Develop a computer program** | **Evidence** |  |
| --- | --- | --- |
| Wrote a program that performs a specific task using a suitable programming language |  | ✓ |
| Set out the program code clearly |  | ✓ |
| Documented the program with comments |  | ✓ |
| Tested and debugged to ensure that it works on a sample of expected cases |  | ✓ |
| **Merit**  **Develop an informed computer program** |  |  |
| Documented the program with variable names and comments that describe code function and behaviour |  | ✓ |
| Following conventions of the chosen programming language |  | ✓ |
| Tested and debugged the program in an organised way to ensure it works on expected and relevant boundary cases |  | ✓ |
| **Excellence**  **Develop a refined computer program** |  |  |
| Ensured the program is a well structured logical solution to the task |  | ✓ |
| Making the program flexible and robust | Quite simple but great structure | - |
| Comprehensively tested and debugged the program | It fails to insert or warn of incorrect data when trying to add things to the db- could have been better tested. | - |

Comments:

Final grades will be decided using professional judgement based on a holistic examination of the evidence provided against the criteria in the Achievement Standard.