Database Assignment

### Purpose and End User of my database

| For anybody who is interested in video games (likely someone who plays them) to be able to view, sort and edit information about different video games of their choice.  This database will allow users to see which games are economical for them- so they can prioritise based on price, genre, storage the game will take up, or the rating of the game out of 10.  They will be able to see these basic details about the games, and if something is of interest to them (e.g. a game with low cost which has a high rating), then they can further research it.  Essentially, the user may wish to look at the existing data in the table which they can utilise to make decisions concerning the games. Alternatively they may want to personalise the database, and edit/add their own data to it (new games, new ratings and details).  This unedited database is sort of like a showcase of what the program is capable of, and the user can build off this and personalise/recompose the whole database (they can add and remove games, update ratings, and display/sort the data in many different useful formats/ways).  **“SEXY” FRAMEWORK:**   * State * Explain * Give Example * Why it's important |
| --- |

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

| **- Usability -**  Usability is an important relevant implication to consider.  It is key that the database is accessible for our user and able to be navigated with relative ease- the purpose of this database is to store our data in an optimised way which also allows the user to access this information without much difficulty.  If our user experience is not taken into consideration, the finished product will likely feel quite low quality in practice.  For instance, instructions should be given clearly throughout the program to guide the user accurately as to what they need to do to have the desired data displayed. Also only having them input what is truly necessary would likely be a nice idea- having all of the boring repetitive code automated will probably make the program much more pleasant to manage for the user.  The interface should be clean and understandable (common should be conventions used), errors should be accounted for and dealt with.  In general, the user should have a smooth experience using our product, and not feel confused or lost.  If the database is difficult to use, then users will have more negative experiences than necessary and simply not be interested in using our database in the future.  **- Functionality -**  Functionality is a very significant relevant implication we need to take into account while creating our databases.  It is important that our database works well consistently and is always displaying correct information, and what the user wants to see. This will be achieved by taking into account what our target audience (the end users) will want/require for the final database and making sure to have all of these key features implemented and working smoothly.  For example, the user will potentially want to find out the viability of buying a game or installing it, so it will be important to allow the user to sort through games and order them by these priorities (like size, cost or rating).  Glitches and bugs/exploits will also need to be dealt with, as these will negatively impact the user experience (this will be done through testing the program on other people, and seeing whether or not they can break it).  If we don’t have the basic and fundamental functionalities working within our databases, then users will not be able to use this database for its purpose (making it essentially useless). It must be able to serve the target audience to a satisfactory standard.  **- Privacy -**  Another relevant implication which is important to consider is privacy.  It is essential that our user’s privacy is protected while using our product- data we collect from the user should be stored securely to make sure that the user is safe and is not vulnerable while using our program.  For instance, with my database, I don’t believe any user information should be required for the functionality of my database, so I will simply collect no data on them. If it was necessary for information to be collected, I would probably want to put in place security measures like passwords, or encrypt user data.  If a user’s information is not protected, this program will be unsafe to use, and thus nobody will want to use it.  It is key that the user's personal data is looked after properly and that we are taking into consideration whether or not personal information is being shared/displayed anywhere (in this case, I don’t believe it should be as much of a concern though). |
| --- |

### Database Design- Your Entity Relationship Diagram.

| [*Link to my lucid charts diagram*](https://lucid.app/lucidchart/171afaba-30f6-4496-b7d3-643dd5b3273e/edit?invitationId=inv_4c2efba7-780c-4f3e-a6d5-da58f642bb46&page=0_0#) |
| --- |

### 

### Database Testing Table: SQL Statements

| **Purpose** | **SQL Statement** | **Result Success?** |
| --- | --- | --- |
| Joins all the tables together, only printing out names/labels (as ID numbers are not relevant to the user). | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID; | ✅ |
| Lists the name of all games present in the table. Allows the user to see what games are in the database, without the screen being as full/cluttered with all of the other details (which they may not need to know, for example maybe they are just checking to see whether the database already contains a game which they are interested in, or if they should add it instead). | SELECT Name FROM Games; | ✅ |
| Allows the user to add a new game to the database, with information about its rating, size, price, and genre. | INSERT INTO Games(Name, Rating, Size\_ID, Price\_ID, Genre\_ID) VALUES (?,?,?,?,?) | ✅ |
| Allows users to delete games from the games table by name. | DELETE FROM Games WHERE Name = ? | ✅ |
| Allows users to update the rating of any existing game in the table. | UPDATE Games SET Rating = ? WHERE Name = ? | ✅ |
| Rank games by price to see which games are most viable to purchase depending on your budget. | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID ORDER BY Price\_ID; | ✅ |
| Rank games by size to see which games are most viable to download depending on the space you have available on your hard drive. | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID ORDER BY Size\_ID; | ✅ |
| Rank games by rating to see which games are the best (based on personal opinion, or the user could add their own data- e.g. official user ratings from steam or another website). | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID ORDER BY Rating; | ✅ |
| Sort games by genre to see which games are more viable to play based on your preference of gameplay. | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID ORDER BY Genre\_ID; | ✅ |
| Find only games less than or equal to a chosen price so that a user can see only games within a chosen budget range. Similar to the order/sort query, but a little more refined and complex, so that the user can isolate games which are viable based on their budget. | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID WHERE Price\_ID <= ?; | ✅ |
| Find only games less than or equal to a chosen size so that a user can see only games within a chosen size range (again, so that the user can isolate games which are viable based on the available space they have on their PC) | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID WHERE Size\_ID <= ?; | ✅ |
| Find only games greater than or equal to a chosen rating so that a user can see only games above a chosen rating (allows the user to isolate and consider games which are rated highly, and ignore the ones with lower ratings). | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID WHERE Rating >= ?; | ✅ |
| Find only games in a chosen genre, so that the user can view these games separately from the others (if they are only interested in a specific genre, this allows them to isolate games based on that). | SELECT Games.Name, Games.Rating, Genres.Genre, Prices.Price, Sizes.Size FROM Games JOIN Genres ON Games.Genre\_ID=Genres.ID JOIN Prices ON Games.Price\_ID=Prices.ID JOIN Sizes ON Games.Size\_ID=Sizes.ID WHERE Genre\_ID == ?; | ✅ |

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

| I have considered and met all relevant implications I identified.  **- Usability -**  *I have met the* ***usability*** *relevant implication by making sure that my program is as intuitive and clearly instructive as possible, by displaying only what is relevant to the user wherever I can, and by taking the user into account when designing the program and such (making it as simple and easy to navigate/view/edit my database as possible).*  *Rather than thinking of this as a programming assignment where I was trying to display technical skills, I needed to move my focus to the intended purpose of my program. A user wants to have a program which allows them to interact with a list of games and attributes about them (view and edit information)- so what could I do to make this as usable and useful and useful as possible.*  *For example, labels are clear and I tried to make everything comprehensible for a more general audience - like rather than having an option labelled “join tables”, I would call it “show more detail/information”. Or as another example, when displaying the games table instead of showing all of the ID numbers (size ID, price ID, genre ID), I instead show the actual genre names and size/price values that those ID’s refer to, as the foreign keys and primary keys by themselves will mean nothing to the user and provide no useful information.*  *These technical labels, ID numbers, optimisations and such are significant to a computer (and thus significant to me, the programmer), however they do not matter to the user.*  *These steps were taken, because as discussed earlier, the purpose of the program is for someone interested in viewing data about video games, and people want something which is intuitive to use, with no unnecessary complications.*  **- Functionality -**  *I have met the* ***functionality*** *relevant implication by making sure that there are lots of useful features/functionalities within my database program, making sure that all functions within the code work smoothly to execute their respective queries and making sure that the program is free of any impactful bugs or glitches.*  *For example, with my database you can: display the games table, list the games present in it, add games to the table, remove games, update game ratings, order games by chosen attributes (e.g. size, price, rating) and select/isolate them based on chosen criteria as well.*  *The reason for this is that the program must have enough functionalities/features to be useful/relevant to our end user, and I wanted the user to be able to do everything which I thought might be helpful for them. Referring back to the purpose of the program: I wanted the user to be able to edit the table (hence allowing them to add, remove and update the games and the information about them), and to organise/prioritise games based on criteria relevant to them (e.g. if they want a cheap games), which is why there is functionality to order and isolate games based on user specific attributes.*  **- Privacy -**  *I have met the* ***privacy*** *relevant implication by making sure no personal information is stored (because I don’t believe this is necessary in order for my program to function). If any data would need to be collected, it would be kept private/secure, but in this case it is not a concern.*  *When using my database you enter no personal information, and no information is taken other than what you enter into the table (information about games)- because it is not necessary for the program to receive anything else.*  *The reason why this is important to consider, is because if the user’s privacy isn’t protected, it would be unsafe and not ideal to use this program, and it would no longer be viable for the end user (defeating the purpose). It is very important that the user’s safety and privacy is valued, as nobody wants to use a program that doesn’t protect their data, and if anything harmful happens as a result of your failure to keep their privacy safe, then it is your responsibility (because you failed to take necessary precautions).*  *Essentially I address this as being a very relevant thing to take into account, however it is not necessarily a concern here, as I have no need to collect user information and do anything with it.* |
| --- |

### 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.

| This program improved a huge amount throughout my development process, as I iterated over it many times- I added much more functionality, heavily improved usability (essentially overhauled it), and made the general aesthetics/polish much better as well towards the end.  As an example, you can see a screenshot below of the first main function of the database- showing the games and the details associated with them in a simple, clean table format *(second photo down)*.  Previously it was a messy group of numbers (foreign and primary keys) that told the database the genre/price/size of each game, but which told the user nothing, and it looked awful *(first photo down)*.  Once I remembered the purpose of this database and its target audience, I was able to significantly improve this usability aspect, only including what is relevant/necessary for the user to see, and streamlining the process as much as I could.  **What it looked like at first (with no formatting):**    **What I got it to look like:**    At the beginning, the functionality aspect of my database was lacking a little in my opinion- you could add and remove games, update game ratings, and show the table, but I wanted more, and I wanted to improve these aspects as well. I had to add features to bulletproof my code- for example I made my own custom function to be used whenever an integer value was needed, so that if the user accidentally (or purposely) entered a non-integer character, it would simply request the user enter a valid integer until one was entered, rather than crashing. I also made a function to check what the upper and lower bounds (highest and lowest possible values) for each input were (e.g. to prevent the user from entering a 100/10 rating, or to try and sort by a genre ID of 4, which doesn’t exist).  Aspects such as this were added to make my program more robust and stable- as upon testing it myself and having others test it, I could see many problems that the user could run into, and many problems the user could cause (myself and some classmates would go through the process of trying to break it in as many ways we could, so that I could see which aspects of functionality and usability I had to improve on for the final product).  On top of these improvements to my program’s robustness, I also improved usability around these functions, with less technical and more understandable labelling, nicer formatting (using line breaks and symbols to separate/break up large amounts of text), and by making sure the user had proper instructions on what to do (all necessary information to navigate and use my database).  For example with the prompts to enter information about a game you were adding to the table- previously it would ask you to “Enter a size ID of 1, 2, or 3” without any other information or context. This was alright for prototyping purposes, but realistically, the user (and myself) do not want to be working with numbers that have no objective meaning. The nice thing about entering numbers over words is that you have to type a lot less, which is why I included them, but they needed more detail and instruction in order to be usable. So instead I would print out something more like “Please enter the size ID of the new game- 1 is small (0-10gb)- 2 is medium (10-30gb)... ” and so on. It could certainly be made more pretty and more concise, but the importance to me was in having something that someone could reliably use, without too much trouble or frustration.  Here are all of the key functionalities/abilities of my program:    And here are some more of those in action:  **Add games:**    As we can see, our new game and its chosen details/data has been successfully added to the table!    **Delete games:**    **Update game rating:**    **Order games (by chosen criteria/attributes):**    **Select/isolate games (by chosen criteria/attributes):**    **Invalid user input handling:**    **The python code behind my program:**  Much of my code was at first very unorganised and messy, but throughout the development process I organised different functionalities into separate functions to avoid duplication of code- allowing for better optimisation and better readability.  For example, the function below takes two user inputs- the name of the game the user wants to change the rating of, and then the new rating they want to apply to it. It is designed in a robust and safe manner, so that if the user doesn’t input the correct thing (or if they try to enter some destructive SQL code), the program will not crash or be harmed by this (the use of the question marks is key here- it allows the program to check whether what the user inputted is a valid/safe query).    This function is what I use every time I need the user to input a number. If the user does not input a valid integer, instead of having the program crash, this function is able to ask for input again until the user enters a valid integer.    This is another input sanitisation function (a function which makes sure that what the user is inputting won’t disrupt my program). This one accepts custom values which determine how big or small the numbers the user is allowed to input are (upper and lower bounds). For instance (as discussed earlier), it will make sure the user cannot enter a rating of 100 out of 10.    Here are all three functions being used together in the main *while loop*:  The user is first asked to input the name of their chosen game, then I use my custom integer function to safely ask for their new chosen rating. My program then checks to make sure that the rating entered is not below 0 or above 10, and if all checks pass, then we commit the user’s changes and the process is complete!    Overall, I am very happy with the final product of my program, and while it could certainly be prettier and more aesthetically pleasing (e.g. with the use of flask), I think that it is functional and usable enough that my target audience could reliably use my program for its intended purpose (organising and observing information about games either for fun or to help people make decisions about these games). I am proud of how far the program has come as I have been developing it, and I have learned a lot along the way- especially about the importance of usability and making your program robust to misinputs. |
| --- |

**Teacher Checklist:**

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

| 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 |  | Y |
| applying appropriate data integrity and testing procedures |  | Y |
| describing relevant implications. |  | Y |
| Merit- Develop an informed digital outcome to manage data |  |  |
| using information from testing procedures to improve the quality and functionality of the outcome |  | Y |
| structuring, organising and querying the data logically |  | Y |
| addressing relevant implications. |  | Y |
| Excellence- Develop a refined digital outcome to manage data |  |  |
| iterative improvement throughout the development and testing process | Love the sort functionality | Y |
| presenting the data effectively for the purpose and to meet end-user requirements. | Nice presentation | Y |

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

**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 |  | Y |
| Set out the program code clearly |  | Y |
| Documented the program with comments |  | Y |
| Tested and debugged to ensure that it works on a sample of expected cases |  | Y |
| **Merit**  **Develop an informed computer program** |  |  |
| Documented the program with variable names and comments that describe code function and behaviour |  | Y |
| Following conventions of the chosen programming language |  | Y |
| Tested and debugged the program in an organised way to ensure it works on expected and relevant boundary cases |  | Y |
| **Excellence**  **Develop a refined computer program** |  |  |
| Ensured the program is a well structured logical solution to the task |  | Y |
| Making the program flexible and robust |  | Y |
| Comprehensively tested and debugged the program |  | Y |

Comments: Unbreakable- would have like more comments but it’s very well structured, logical and functional.