Unit XVIII Assignment I

By Nathan Windisch

Task I: Relational Databases

PI: Business Information, Entities, Attributes and Data Validation

Business Information

Business Information is data such as client information, stock information, share information and more. All of these individual pieces of data should be stored in a database to allow easier access to the system, resulting in faster data transference and quicker response times. Databases come in many forms and use different languages such as `SQL` and `dBase`, and can be used in many different ways. The business information that is saved should be encrypted, to prevent unauthorized access of the data, which may be considered sensitive. Encryption of this data is not just a good practice as far as customer relations go, as they know that their data is safe, but it also can be a legal requirement in some countries, meaning that the business is liable if any data is lost or stolen.

Entities

Entities are points of data that can be found within a database. Entities are inherently linked to their appropriate entities and are considered to be the unique identifier of the data. Entities can be a single person, place, item of stock or any other "thing" which has properties. Entities can be related to other Entities with a system called normalisation, which is where entities can be connected to one another. Business data can be represented using entities using

Attributes

Attributes are single pieces of data within a database that allows different types of data to be stored. This can include, but is not limited to, certain types of business data such as client information, stock information and share information. All of these data types will have many different entities within them, such as customer first names, dates of births of employees, any allergies that managers may have and more. All of these entities will probably be just one type of data, such as an integer or a character. Most of these primitive data types should already be known as they can be seen in many common programming languages. Other, more specific primitive data types can be found within some database languages, but the most common one is varchar that is found within SQL. While all the previous data types can only contain one type of character (although strings can store integers in a text based format), a varchar can store both characters and integers in a similar format to a string in other programming languages. This means that you can store numbers as text, if the context requires it. It is important to use the correct data types when storing different pieces of information, due to the fact that it can take up less space and it can also mean that the data can be read easier and the data can be accessed and sorted faster. Business information can be stored as attributes, such as a person’s name, date of birth or address.

Entity Relationship

Entity Relationship is a way of creating diagrams that show how entities can be linked together. This allows database designers to be able to visualise how their database will work, allowing easier access to the system along with easier database population and mapping for future data being added to the system.

Data Validation

Data Validation is a method of ensuring that all data within the database is valid and accurate. This can be done by performing checks when the data is entered and throwing an exception if an error is made. Another way that this can be achieved is by only allowing certain data types to be allowed to be entered into a field, such as only integers being allowed for an age to be input. This can be expanded upon to ensure that the data that is input into a Date Of Birth field has six numbers split with a slash, such as `08/09/98` for the `8th September 1998`. Data Validation is a huge part of database management as if the data is not validated and the inputs are not sanitized, then hackers can inject malicious code into the system such as `DROP TABLE Customers;` which will result in the database `Customers` being deleted.

PII: Designing a Relational Database

Normalization

Within databases there are three different forms of normalization.

- **UNF**, or **Unnormalized** is when there is lots of data redundancy and can contain many data structures within a single hallmark.
- **1NF**, or **First Normal Form** is when each field in a table does not contain the same type of information. An example of this would be in a customer list where each table would only contain one phone number.
- **2NF**, or **Second Normal Form** is when each field in a table must be a function of the other fields in the table if it is not a determiner of the contents of that field
- **3NF**, or **Third Normal Form** is when there is absolutely no duplicate information within the table. For example, if two tables both require a phone number field, that information would be placed into a separate table, and the two other tables would then information that they want such as the phone number data, via an index field in the newly created phone number table. Any and all change that are made to a phone number will now automatically update and be reflected to all tables that use the phone number table.

Examples

The following are some examples of different relational databases that I have mocked up with items based around gardening:

UNF

ID	Type	Price
01	daisy	07.49
02	iris	02.24
03	lavender	06.87
04	pansy	21.05

1NF

Unnormalized

ID	Price
01	07.49
02	02.24
03	06.87
04	21.05

1NFified

ID	Type
01	daisy
02	iris
03	lavender
04	pansy

2NF

Unnormalized

ID	Client ID	Balance
01	110	96.03
02	420	05.24
03	911	12.62

2NFified

ID	Client ID
01	110
02	420
03	911

Client ID	Balance
110	96.03
420	05.24
911	12.62

3NF

Unnormalized

Flower ID	Bloom Season ID	Literal Season Name	Price
221	001	Spring	05.88
486	002	Summer	09.15
683	001	Spring	35.73

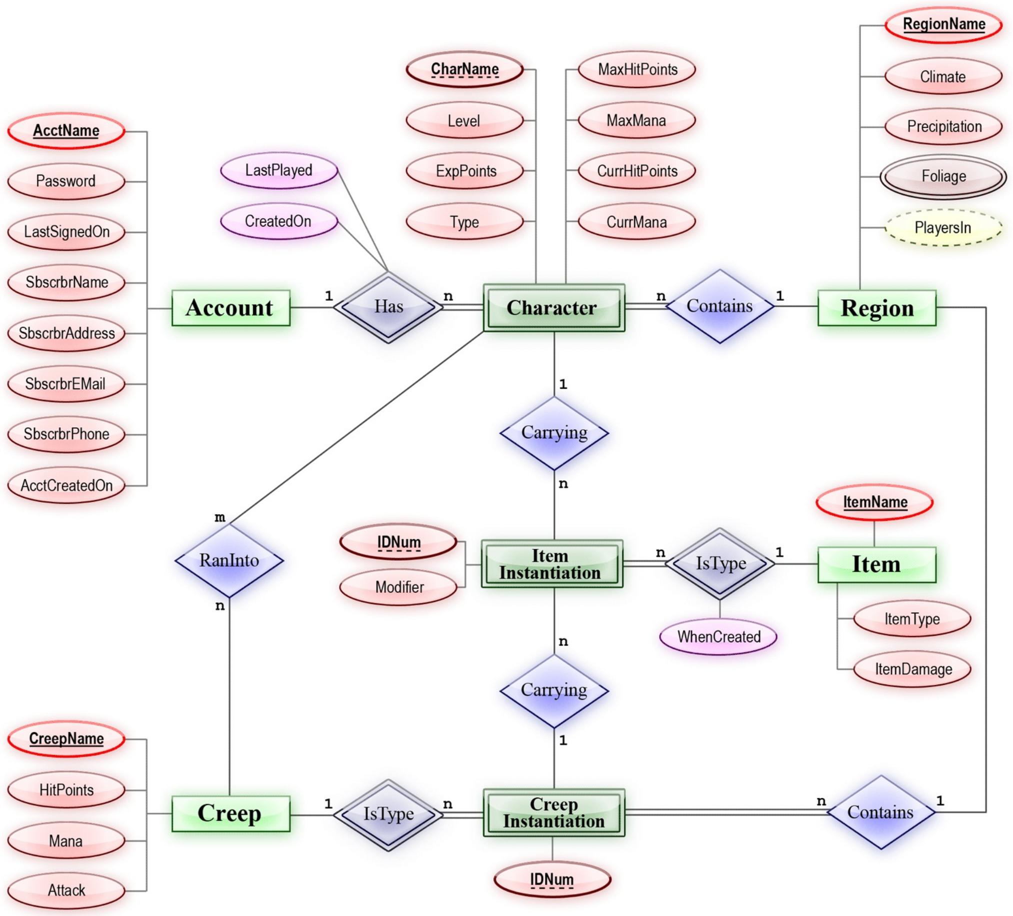
3NFified

Flower ID	Bloom Season ID	Price
221	001	05.88
486	002	09.15
683	001	35.73

Bloom Season ID	Literal Season Name
001	Spring
002	Summer

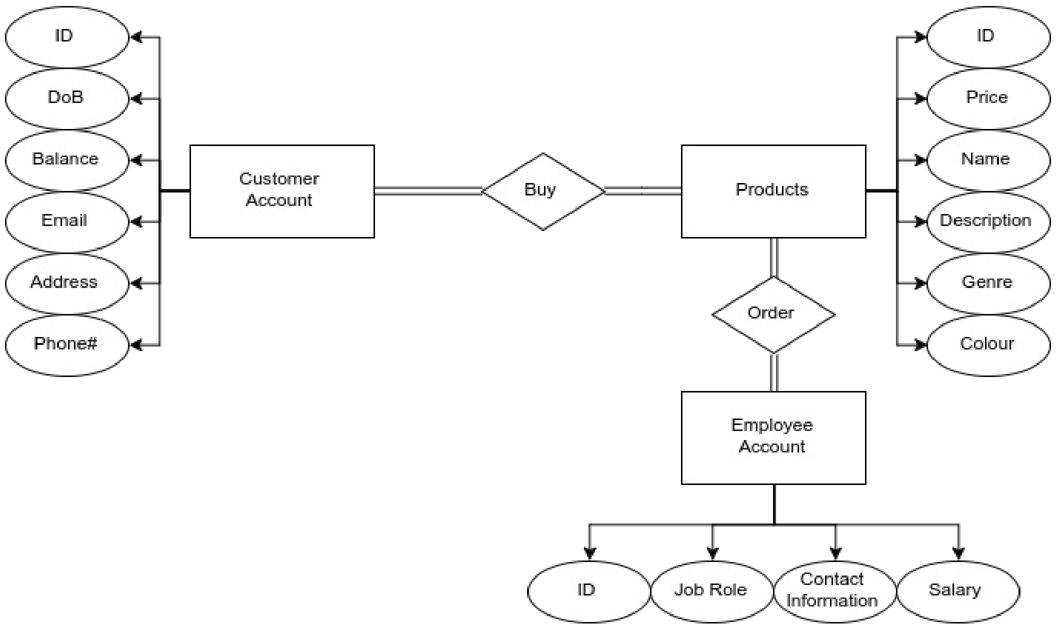
Entity Relationship Diagram

When creating a relational database, an Entity Relationship Diagram must be created in order to define how different parts of the database will fit together. The following is an example from wikimedia.org about an MMORPG:



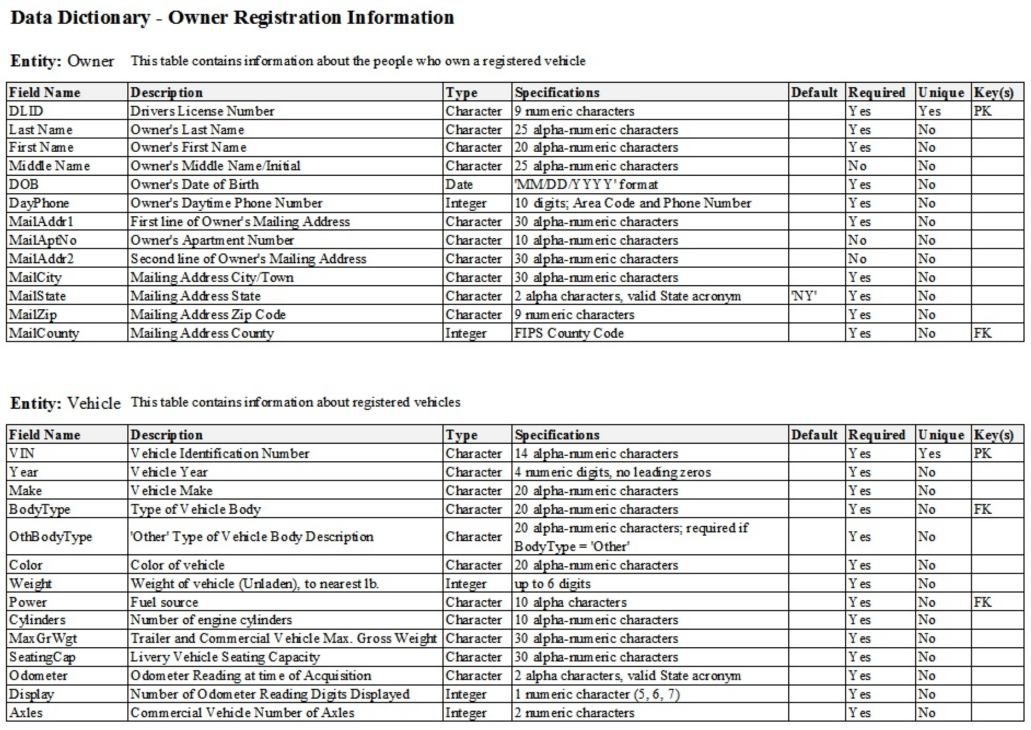
The diagram above shows all of the entities within the game, as shown by the green boxes, and how they are related to one another using the blue diamonds. For example, in relation to the diagram above, one account can have many characters, but each character can only have one account. Each region can contain many characters, but each character can only be in one region. Each of the entities contains values, as shown by the red circles. The primary key, or main defining characteristic, is in a lighter red circle with emboldened and underline text. Again, another example using the diagram above, is that each account has an account name, a password, a date where they last signed on and more, but the account name is the primary key that is used as the unique identifier for each account.

The following image is my Entity Relationship Diagram:



Data Dictionary

Data Dictionaries are a method of laying out data in a way that allows it to be read easily by a human. The formatting should be well sorted and concise in a table or matrix, and would normally contain simple data like the variable name, type of data and size of the field. The following is an example of a data dictionary, also taken from wikimedia.org:



Data Dictionary - Owner Registration Information

Entity: Owner This table contains information about the people who own a registered vehicle

Field Name	Description	Type	Specifications	Default	Required	Unique	Key(s)
DLID	Drivers License Number	Character	9 numeric characters		Yes	Yes	PK
Last Name	Owner's Last Name	Character	25 alpha-numeric characters		Yes	No	
First Name	Owner's First Name	Character	20 alpha-numeric characters		Yes	No	
Middle Name	Owner's Middle Name/Initial	Character	25 alpha-numeric characters		No	No	
DOB	Owner's Date of Birth	Date	'MM/DD/YYYY' format		Yes	No	
DayPhone	Owner's Daytime Phone Number	Integer	10 digits; Area Code and Phone Number		Yes	No	
MailAddr1	First line of Owner's Mailing Address	Character	30 alpha-numeric characters		Yes	No	
MailAptNo	Owner's Apartment Number	Character	10 alpha-numeric characters		No	No	
MailAddr2	Second line of Owner's Mailing Address	Character	30 alpha-numeric characters		No	No	
MailCity	Mailing Address City/Town	Character	30 alpha-numeric characters		Yes	No	
MailState	Mailing Address State	Character	2 alpha characters, valid State acronym	'NY'	Yes	No	
MailZip	Mailing Address Zip Code	Character	9 numeric characters		Yes	No	
MailCounty	Mailing Address County	Integer	FIPS County Code		Yes	No	FK

Entity: Vehicle This table contains information about registered vehicles

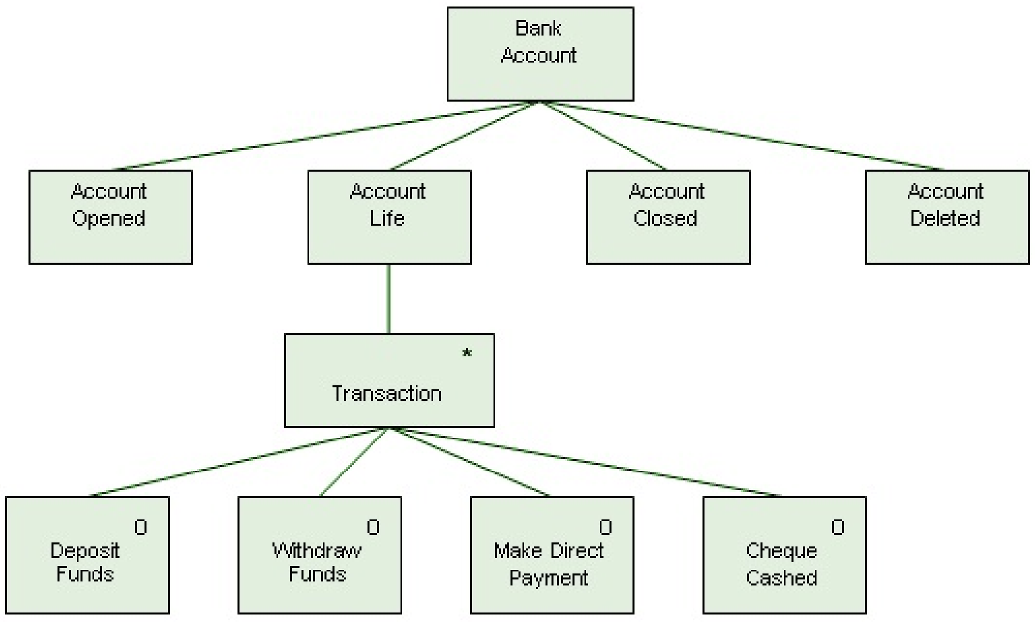
Field Name	Description	Type	Specifications	Default	Required	Unique	Key(s)
VIN	Vehicle Identification Number	Character	14 alpha-numeric characters		Yes	Yes	PK
Year	Vehicle Year	Character	4 numeric digits, no leading zeros		Yes	No	
Make	Vehicle Make	Character	20 alpha-numeric characters		Yes	No	
BodyType	Type of Vehicle Body	Character	20 alpha-numeric characters		Yes	No	FK
OthBodyType	'Other' Type of Vehicle Body Description	Character	20 alpha-numeric characters; required if BodyType = 'Other'		Yes	No	
Color	Color of vehicle	Character	20 alpha-numeric characters		Yes	No	
Weight	Weight of vehicle (Unladen), to nearest lb.	Integer	up to 6 digits		Yes	No	
Power	Fuel source	Character	10 alpha characters		Yes	No	FK
Cylinders	Number of engine cylinders	Character	10 alpha-numeric characters		Yes	No	
MaxGrWgt	Trailer and Commercial Vehicle Max. Gross Weight	Character	30 alpha-numeric characters		Yes	No	
SeatingCap	Livery Vehicle Seating Capacity	Character	30 alpha-numeric characters		Yes	No	
Odometer	Odometer Reading at time of Acquisition	Character	2 alpha characters, valid State acronym		Yes	No	
Display	Number of Odometer Reading Digits Displayed	Integer	1 numeric character (5, 6, 7)		Yes	No	
Axles	Commercial Vehicle Number of Axles	Integer	2 numeric characters		Yes	No	

The following is an example of a data dictionary that I have created for the gardening company **Coastal Corners**:

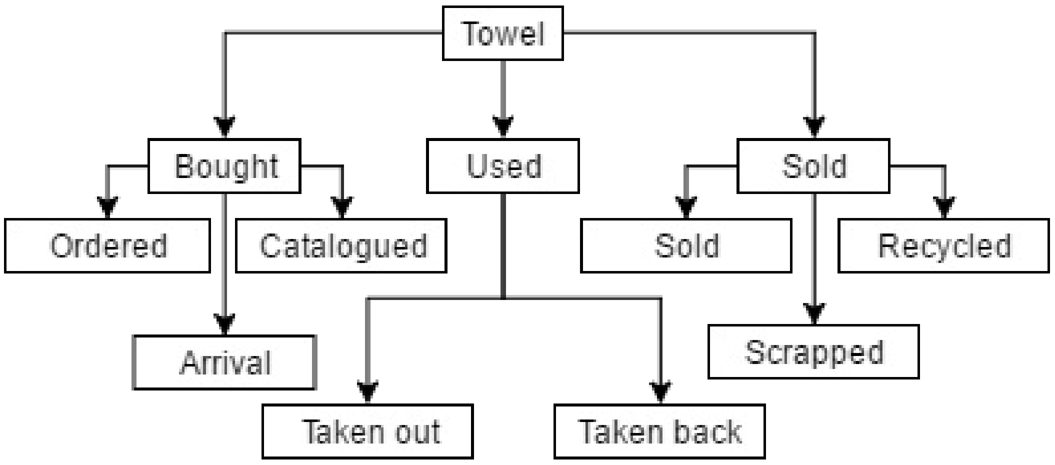
Field Name	Data Type	Maximum Field Length	Description
EmployeeID	UUID	16 Characters	Used as a unique identifier for each employee
EmployeeName	String	64 Characters	Used as a human readable format for identifying customers
EmployeeRole	String	16 Character	Used for the permission system
RoleDescription	String	128 Character	A brief description of their role
RoleWageMin	Double	10 Digits	The minimum amount that a person in that role can be paid
RoleWageMax	Double	10 Digits	The maximum amount that a person in that role can be paid
CustomerID	UUID	16 Characters	used as a unique identifier for each customer
CustomerName	String	64 Characters	Used as a human readable format for identifying customers
CustomerBalance	Double	10 Digits	The amount of credits that the customer has
CustomerAddress	String	64 Characters	The address of the customer
CustomerPostcode	String	8 Characters	The postcode of the customer
CustomerPhoneNumber	Integer	15 Characters	Either the mobile or the landline number of the customer
ItemID	UUID	16 Characters	Used as a unique identifier for each item
ItemName	String	64 Characters	Used as a human readable format for identifying items
ItemPrice	Double	10 Digits	The price of the item
ItemDescription	String	128 Characters	A brief description of the product
ItemTags	ArrayList	64 Characters Per Entry	A list of tags that can be applied to the item

Entity Life History

The life history of an entity is a list of actions that are performed to a entity within it’s lifetime. All changes that are made to the entity are tracked and can be displayed in a flow chart. The following image is from technologyuk.net and is about bank account information:

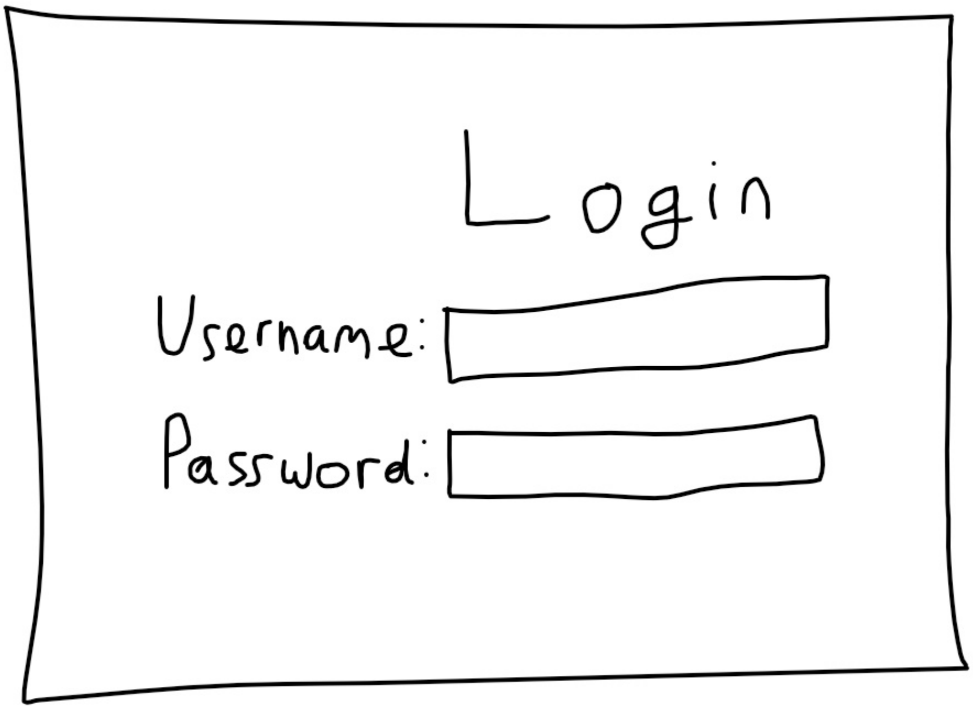


The following is my Entity Life History Diagram for a batch of trowels:

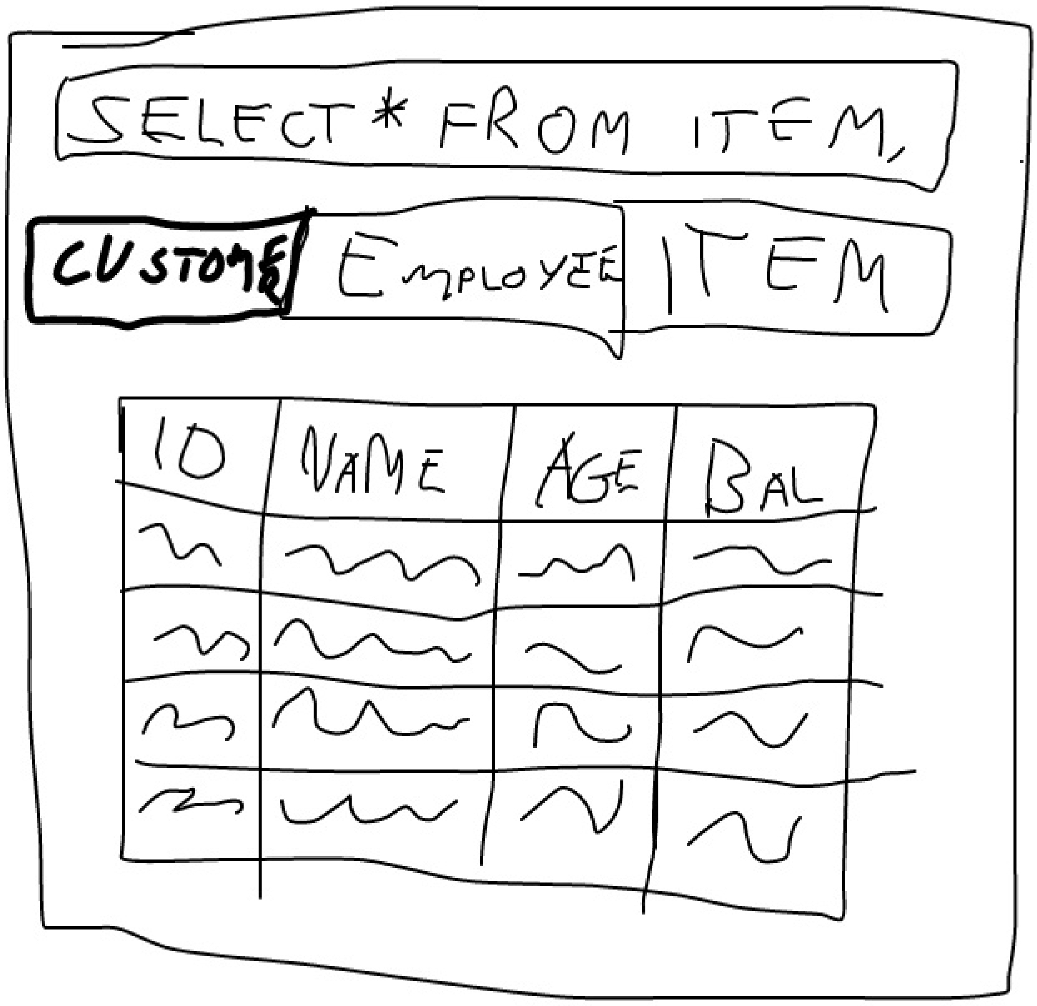


Front End Design

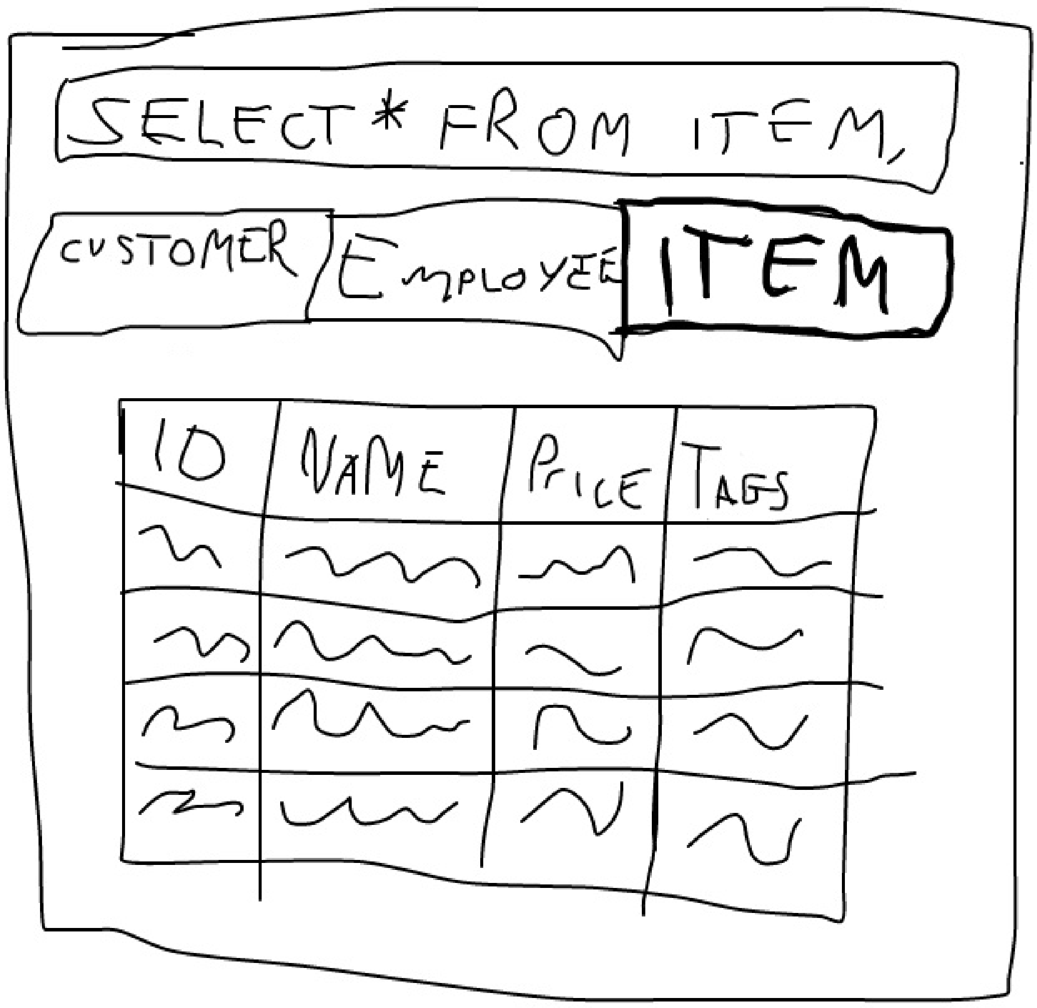
The following are three sketches for front end interfaces that the users will use. This first one will be the login screen:



The second one will be that of the querying of the database after logging in:



The third and final sketch will be the report that is outputted to the user after they query:



MI: Explaining Referential Integrity and Primary & Foreign Keys

How Referential Integrity is Set in a Database

First off, referential integrity is a relational database concept about how all table relationships must be constant and consistent. To put this in more simple terms the `foreign key`, which is a field or fields which uniquely identify either a row in the same or another table, must be similar to the `primary key`, which is the unique identifier for each record. Referential connect databases together by ensuring that the data is properly referenced and verified. Most database creation softwares do this function automatically, as it is important for relational databases to have as it is what inherently makes them relational.

Errors Detected and Fixes

There are two main type of referential integrity errors, the first being when the parent table is being edited. If a primary key value is updated or edited and the rule applies to foreign keys within the child table, then the secondary database will reject the delete or update commands. If you want to ensure that the changes to the primary key values within the parent table are saved and propagated then the child table must be edited first to ensure that all the matching foreign keys are either changed or deleted first.

The second type of referential integrity error is when a foreign key value is created within a child table but there is no matching entry within the parent table. The second database will reject the command that has been issued due to the fact that that value within foreign keys cannot exist without a primary key value in the corresponding table. To fix this issue and change the foreign key value in the child table, the parent table must be edited and a row should be created with a primary key value that matches the foreign key's new value.

Data Recovery Using Multiple Tables

When recovering data from tables, both the primary and secondary keys should be used. This is because the data that is linked from the foreign key from one table, and the primary key of the other. A good way of thinking about this is imagining that the keys are pointers which tell the database where the data is actually stored, rather than it being an actual clone of the data. This method can also work with multiple tables, as long as they are all in 3NF as they allow the database software to just search through the pointers and find the original data.

DI: Avoiding Potential Design & Construction Errors

Inconsistent Naming Schema

One common error in database design is having an inconsistent naming schema. An impact of this is that it will be harder to search for things in the database if the search query is case sensitive. An example of this changing prefixes of data, such as all data retaining to customers, from being FULL CAPS, camelCase and lowercase. The following is a quick table with some values to give a more visual representation of what this would look like:

Name	Value
customer_name	"John Smith"
CUSTOMER_dob	08091998
customer_PostCode	RG67NW

A way to fix this issue is to have a well documented naming schema to ensure that all entries follow a strict standard, and are even rejected if they are incorrect. This leads me into my second potential error: poor documentation.

Poor Documentation

Poor documentation is a large issue as it makes certain employees indispensable. This means that when they leave the company, all of their knowledge of the system will leave with them. This can result in customers not getting the correct orders or information being deleted. A way to solve this issue is to document the way that the database is laid out. An example of some well documented database variables is as follows:

Name	Description
CUSTOMER_name	A string used to store the customer’s name
CUSTOMER_age	A double used to store the customer’s age
ITEM_id	A unique identifier for quickly accessing items from the database; should only be used in the backend

Normalization

Every database within a company should be set to the Third Normal Form, as it is the most efficient and easy to search for. One impact of lower levels of normalization is that searching within the database will be much less efficient due to the fact that more tables will need to be searched. A way to avoid this is to set up the database in such a way that the database is very efficient and easy to search. An example of this will be shown below:

Unnormalized

Flower ID	Bloom Season ID	Literal Season Name	Price
221	001	Spring	05.88
486	002	Summer	09.15
683	001	Spring	35.73

3NFified

Flower ID	Bloom Season ID	Price
221	001	05.88
486	002	09.15
683	001	35.73

Bloom Season ID	Literal Season Name
001	Spring
002	Summer

Deleting

The final issue that can be made when making a database is when you want to delete data. Deleting data permanently is usually a bad thing, as more often than not the data will be needed later down the line. This means that data will be deleted permanently without the ability to recover it later. A real world example of this is when a customer deactivates their account with the company so the data is hard deleted, only for them to want to resubscribe a few months later and having to reenter all of their data again. The solution to this is to set the row to be inactive so that it will not show up in searches, but the data can be reactivated later with ease.