**Databases Coursework Design Document**

C:\Users\40173513\Documents\Databases\Coursework\UML Diagram.png

**Reasons for using the types:**

* **Name –** Grouping the title, first name and surname together just to make the design much nicer, it makes sense to group these 3 values together and it makes for a very robust type that can be reused over and over again and inherited from in many situations.
* **Job –** Once again, splitting up the design into more types to make it much more readable, and these 4 values make sense to be together. This type will be used for the final employee type, which will make the employee type look much cleaner and ease up on the overall design of the system. It also has a reference to **bID** which links employee and the branch tables.
* **Address –** Used for multiple other types, such as Branch and Person. This makes the system way more readable and robust as this type is used twice in the system and due to 2 other types inheriting from person, it is actually 4 types that use address. This type is definitely the one that is used the most and it makes other types more organised and results in less redundant data.
* **Phone -**  Phone is an interesting type as it has a value for the home number and a nested table for mobile numbers. I decided to group these 2 into a type because this will also make the system more concise and this type can be reused for both customer and employee. A nested table was used so that the types that use this will not have to worry about running out of space for mobile numbers and honestly, some people have multiple mobile phones in real life and it would be nice to design the system in such a way that those people will be allowed to have all their mobile phone number sin while other people who have just 1 mobile phone number won’t be hindered by this.
* **Person –** This type is the backbone of the Customer and Employee types. This contains all the necessary values to be used in those 2 types. I have decided to make this type to use inheritance in the system to reduce the amount of redundant variables in the design. This makes the design look less cluttered.
* **Customer –** It inherits from the Person type and adds a custID to serve as a primary key. This type will be used for the Customer table.
* **Employee -**  This type inherits from Person and adds a empID to serve as a primary key. It also will have a reference inside itself to have **supervisorID**. I did it this way to make it super simple to access the supervisor’s data whenever we need it. This type also has a member function which will be explained later in this document.
* **Branch -**  Uses Address type and a phone variable that is not the phone type as phone type also has a nested table in it that is unnecessary for branch so there would be no benefit from just using phone type for this. Used for the final branch table.
* **CustomerAccount -**  Used for CustomerAccount table. Contains a reference to both the customer table and the account table.
* **Account –** Used for the account table, doesn’t use any other types for its values but it does have a reference to Branch as **bID** links both the tables.

**Datatypes used**

One of the datatypes used is Varchar2, which was used for majority of the elements as a lot of them were of a string type and they were not needed for any calculations so they can be stored as a string. Varchar2 was used over varchar as it is the recommended by oracle and it stores more bytes than varchar. Phone numbers are stored as varchar2s as they require a 0 to be stored at the front and the Number datatype does not allow for a 0 to be placed at the front.

Another datatype that was used in the Number datatype, this was used for any IDs and any balance information as it allows us to perform mathematical operations on these elements which may be required.

The final datatype used is the Date datatype, this one is used for any dates that are stored in the system. This allows us to do any calculations concerning dates, for example we can find the number of years an employee worked at a branch.

**References**

I have used references instead of foreign keys as they simplify queries much more. Using references means that I do not have to use any type of joins to make each table interact with each other. It really makes queries much easier and nicer. References also allow for storing of tuples in a single attribute of the table so it is easy to access any necessary information from the referenced tuple.

**Using nested tables over varrays**

Varrays are ordered lists which are stored in line (in the table itself). I decided to use nested tables over varrays because nested tables have many more benefits in this problem. One reason I decided to pick nested tables over varrays is that we can easily extend a nested table, which means that we can have an infinite number of phone numbers for customers and employees, whereas varrays are limited by an upper bound. Another reason why I decided to pick nested tables over arrays is the fact that when you want to access an element in a varray, you have to retrieve all of the elements in the varray because a varray is stored as a single object, which is not the case in a nested table, where you can easily access a single element in a nested table.

**Using methods**

In this implementation I decided to use 2 methods, calculateYears and award. calculateYears takes in 2 dates, namely startDate and endDate and calculates the number of years the employee has worked for. Award takes in the calculated years number and takes in a number of supervised employees to decide what award each employee deserves, if none of the requirements are not reached then the return value will be a string saying null which will make sure this employee is not displayed in the output. I had a 3rd function which calculates the number of supervised employees but finally decided that it would be much easier to find that out in the actual statement than create a new method for this.

**Constraints used**

The constraints that I used mostly are just not null constraints as its bad to have an actual database with null values scattered about, it makes the database not accurate. For example, not having any address information, if anything happens then the person using the database would not be able to know where to send help. On the other hand, title is not crucial as it’s just a little honorific so not having it in every tuple won’t hinder the database at all, also some people do not have any phone numbers so it would be good to accommodate these people too.

The other constraint that I used is primary key, it works in the same way as the primary key in ER systems and it allows me to have the functionality of unique and not null in one constraint rather than using two constraints for that.

National Insurance number is the only value that has the unique constraint as a national insurance number must be a unique number.

I have also included the special requirements such as position having a check constraint to make sure the position should be either a Manager, Head, Leader, Accountant or Cashier. I also added a check constraint to the type of account to make sure its only a savings account or a current account.