July 30, 2020

Juan alvarez

44340725

Software Engineering

SET09107: Advanced Database Systems  
2019/2020

Coursework

Contents

[Task 1 2](#_Toc47097352)

[Task 2 2](#_Toc47097353)

[2.1 Person type 2](#_Toc47097354)

[2.2 Address type 2](#_Toc47097355)

[2.3 Phone type 2](#_Toc47097356)

[2.4 Nested phone type 3](#_Toc47097357)

[2.5 Employee type 3](#_Toc47097358)

[2.6 Client type 3](#_Toc47097359)

[2.7 Branch type 3](#_Toc47097360)

[2.8 Account type 3](#_Toc47097361)

[2.9 Client-Account type 3](#_Toc47097362)

[2.10 Tables 3](#_Toc47097363)

[2.10.1 Branch table 3](#_Toc47097364)

[2.10.2 Account table 4](#_Toc47097365)

[2.10.3 Customer table 4](#_Toc47097366)

[2.10.4 Customer-Account table 4](#_Toc47097367)

[2.10.5 Employee table 4](#_Toc47097368)

[3 Implementing Database 5](#_Toc47097369)

[3.1 Populating the Database 8](#_Toc47097370)

[4 Providing SQL statements 8](#_Toc47097371)

[4.1 A 8](#_Toc47097372)

[4.2 B 9](#_Toc47097373)

[4.3 C 9](#_Toc47097374)

[4.4 D 10](#_Toc47097375)

[4.5 E 10](#_Toc47097376)

[4.6 F 11](#_Toc47097377)

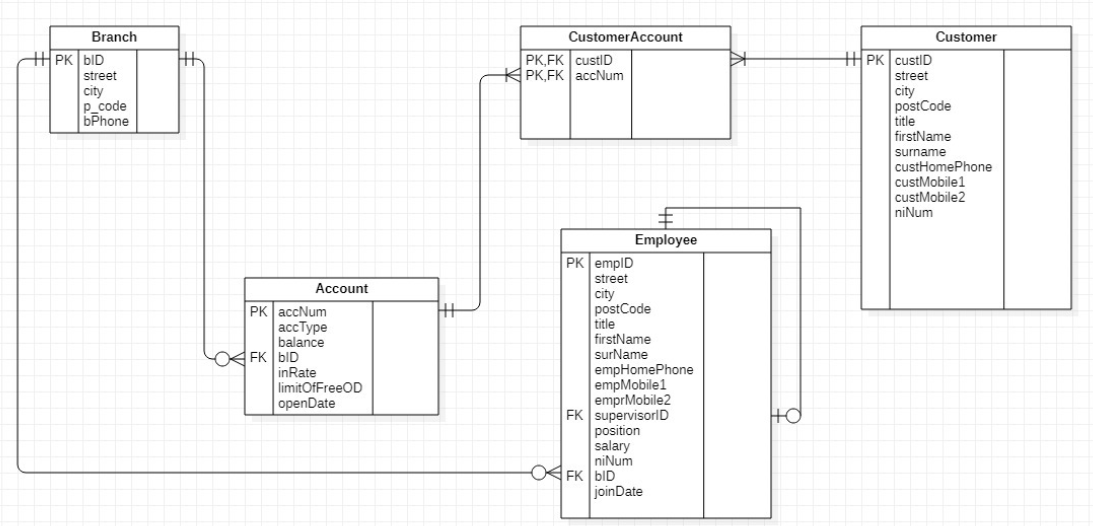
[4.7 G 12](#_Toc47097378)

[4.8 H 13](#_Toc47097379)

[5 Object-Relational vs Relational model 14](#_Toc47097380)

[6 Deleting the database 14](#_Toc47097381)

# Task 1



# Task 2

In order to have better control and maintain the principle of single responsibility, it has been decided to create each of the types separately and not to add them together. That is, in the case of the person type, a super type could be created that has as data: person type, address type and telephone type; but in this way the information of the people would have been coupled with information that perhaps is not always necessary. And it is for this reason that they have been rejected and have not been added in a super type and it has been decided that all the types that need the data will use them as required.

## 2.1 Person type

As a first step for the redesign of the database, the person type has been created to be able to frame and generalize the data [title, names, surnames, national insurance number]. In this way they can be reused homogeneously in as many other types of data as necessary. Also taken into account for the creation of this type of data is the ability to simultaneously modify all types of data that inherit from the "mother" type, for example in the case of needing to increase or modify any of the data of this type, you can automatically replicate it in the inheritance.

## 2.2 Address type

As in the previous case, the address type has been created with the fields [street, city, postal code] in order to homogenize the fields needed for the storage of addresses, so that as soon as an address needs to be stored it will always be the same fields; it also has the advantage of adding and modifying fields, that is, when the type is altered in any way it can be propagated everywhere.

Where it is being referenced: The address type is used in office type, employee type and customer type

## 2.3 Phone type

In order to have a more general database, the phone type has been created with the fields [type, number] so that in this way you can store the phones of the offices, customers and employees in a more homogeneous way and without having to create a column for each time you need to store a new type of phone.

## 2.4 Nested phone type

This type of data has been created to be able to give solution together with the previous point to the storage of the telephones in the database. The use of vArray data type has been valued; but being of fixed size it would not give so much flexibility in the future when some other type of phone needs to be stored. For example, in the future you may need to store the fax phone of a branch office or the mobile phones with WhatsApp/Telegram or similar to be able to contact them by other means.

## 2.5 Employee type

This type of data inherits from the data type Person, abstracting from the data that a person may have and only declaring the data that is specific to an employee. It also includes within its fields the type Address, the nested telephone table, the corresponding supervisor, the position held, the year's salary, the office where he/she works and the date on which he/she joined the bank.

## 2.6 Client type

Again, this type of data inherits from the type Person, it also abstracts from the fields that this type may have and declares those that are necessary to store the information of a client: address, phone’s nested table.

## 2.7 Branch type

This type of data is created to be able to store the information of the bank's branches. Among its fields are: the branch unique identifier, the Address type and the phone’s nested table.

## 2.8 Account type

The Account type has been created to be able to store the data concerning the clients' accounts, within the fields are: the unique account identifier, the account type, its balance, reference to the associated branch, its interest rate and its overdraft limit.

## 2.9 Client-Account type

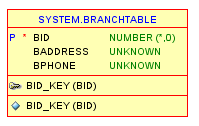
Finally, for the proposed model I have created the type Client-Account in which all the necessary data is related to be able to determine which client(s) have what account(s). The only fields it has are: the reference to the customer type and the reference to the account type.

## 2.10 Tables

The design for the tables created using the above mentioned types are:

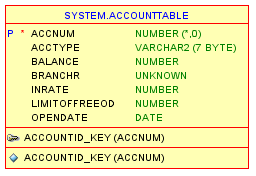
### 2.10.1 Branch table

A table to store the bank’s branch details.



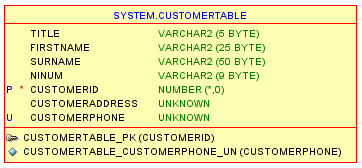
### 2.10.2 Account table

All accounts opened in the bank will be store in this table independently of who opened it.



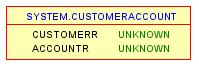
### 2.10.3 Customer table

All the customers will be stored in this table, it has a phone table nested on it using the phone type and nested phone type.



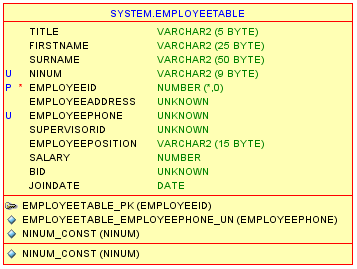
### 2.10.4 Customer-Account table

This table joins customers with their respective bank accounts.



### 2.10.5 Employee table

This table is used to store the bank’s employees.



# 3 Implementing Database

The implementation of the database is included on the file called DBcreating.sql as requested.

A coy of it is found here:

CREATE OR REPLACE TYPE addressType AS OBJECT  
(  
street varchar2(20),  
city varchar2(20),  
postal\_code varchar2(8)  
)  
final  
/

CREATE OR REPLACE TYPE phoneType AS OBJECT  
(  
phoneType varchar2(10),  
phoneNumber varchar2(15)  
)  
final  
/

CREATE OR REPLACE TYPE branchType AS OBJECT  
(  
bID integer,  
bAddress addressType,  
bPhone phoneType  
)  
/

CREATE TABLE branchTable OF branchType  
/

ALTER TABLE branchTable  
ADD(CONSTRAINT bID\_key PRIMARY KEY(bID))  
/

CREATE OR REPLACE TYPE accountType AS OBJECT  
(  
accNum integer,  
accType varchar2(7),  
balance number,  
branchR REF branchType,  
inRate number,  
limitOfFreeOD number,  
openDate date,  
CONSTRUCTOR FUNCTION accountType(a\_accNum integer, a\_accType varchar2,  
a\_balance NUMBER, a\_openDate date) RETURN SELF AS RESULT,  
MEMBER PROCEDURE insert\_records,  
MEMBER PROCEDURE display\_records);  
/

CREATE TABLE accountTable OF accountType;  
/

ALTER TABLE accountable  
ADD(CONSTRAINT accountID\_key PRIMARY KEY(accNum));  
/

CREATE OR REPLACE TYPE BODY accountType AS CONSTRUCTOR FUNCTION accountType(a\_accNum integer, a\_accType VARCHAR2, a\_balance NUMBER, a\_openDate date)  
RETURN SELF AS RESULT  
IS  
BEGIN  
Dbms\_output.put\_line('Constructor fired..');  
SELF.accNum:=a\_accNum;  
SELF.accType:=a\_accType;  
SELF.balance:=a\_balance;  
SELF.inRate:=0.01;  
SELF.limitOfFreeOD:=0.02;  
SELF.openDate:=a\_openDate;  
RETURN;  
END;  
MEMBER PROCEDURE insert\_records  
IS  
BEGIN  
INSERT INTO accountable  
VALUES(accNum, accType, balance, branchR, inRate, limitOfFreeOD, openDate);  
END;  
MEMBER PROCEDURE display\_records  
IS  
BEGIN  
Dbms\_output.put\_line('Account Number:'||accNum);  
Dbms\_output.put\_line('Account Type:'||accType);  
Dbms\_output.put\_line('Balance:'||balance);  
Dbms\_output.put\_line('In Rate:'||inRate);  
Dbms\_output.put\_line('Limit Of Free OverDraft:'||limitOfFreeOD);  
Dbms\_output.put\_line('Open Date:'||openDate);  
END;  
END;  
/

CREATE OR REPLACE TYPE peopleType AS OBJECT  
(  
title varchar2(5),  
firstName varchar2(25),  
surName varchar2(50),  
niNum varchar2(9)  
)  
not final  
/

CREATE OR REPLACE TYPE phoneNested AS TABLE OF phoneType  
/

CREATE OR REPLACE TYPE customerType UNDER peopleType  
(  
customerID integer,  
customerAddress addressType,  
customerPhone phoneNested)  
/

CREATE TABLE customerTable of customerType  
(  
PRIMARY KEY(customerID)  
) NESTED TABLE customerPhone STORE AS phoneNestedTable  
/

CREATE OR REPLACE TYPE employeeType UNDER peopleType  
(  
employeeID integer,  
employeeAddress addressType,  
employeePhone phoneNested,  
supervisorId REF employeeType,  
employeePosition varchar2(15),  
salary number,  
bID REF branchType,  
joinDate date  
)  
/

CREATE TABLE employeeTable OF employeeType  
(  
PRIMARY KEY(employeeID),  
CONSTRAINT niNum\_Const UNIQUE(niNum)  
)NESTED TABLE employeePhone STORE AS phoneEmpNestedTable  
/

CREATE TYPE customerAccountType AS OBJECT  
(  
customerR REF customerType,  
accountR REF accountType  
)  
/

CREATE TABLE customerAccount  
(  
customerR REF customerType SCOPE IS customerTable,  
accountR REF accountType SCOPE IS accountable  
)  
/

## 3.1 Populating the Database

The code to populate the database is too long to include in this document, so instead I just mention that it has been included in the submission pack with the name DBPopulationg.sql as requested.

# 4 Providing SQL statements

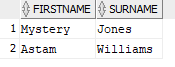
All statements bellow are as well included in the separate file called answerToTask4.sql.

## 4.1 A

Find employees whose first name includes the string “st” and live in Edinburgh, displaying their full names:

SELECT e.firstName, e.surName  
FROM employeeTable e  
WHERE e.firstName LIKE '%st%'  
AND e.employeeAddress.city = 'Edinburgh'  
/

Result:

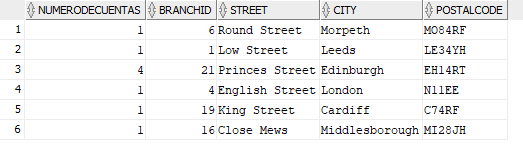


## 4.2 B

Find the number of saving accounts at each branch, displaying the number and branch’s address.

SELECT NumeroDeCuentas,  
BranchId,  
b.bAddress.street AS Street,  
b.bAddress.City AS City,  
b.bAddress.postal\_code AS PostalCode  
FROM (  
SELECT COUNT(\*) AS NumeroDeCuentas,  
ca.accountR.branchR.bID AS BranchID  
FROM customerAccount ca  
WHERE ca.accountR.accType = 'SAVINGS'  
GROUP BY ca.accountR.branchR.bID  
), branchTable b  
WHERE BranchId = b.bID  
/

Result:

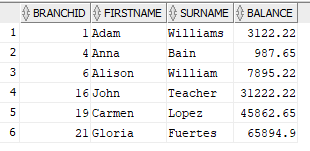


## 4.3 C

At each branch, find customers who have the highest balance in their savings account, displaying the branch ID, their names, and the balance.

SELECT BranchId,  
ac.customerR.firstName AS FirstName,  
ac.customerR.surName AS SurName,  
BalanceMax AS Balance  
FROM (  
SELECT MAX(ca.accountR.balance) AS BalanceMax,  
ca.accountR.branchR.bID AS BranchId  
FROM customerAccount ca  
WHERE ca.accountR.accType = 'SAVINGS'  
GROUP BY ca.accountR.branchR.bID  
), customerAccount ac  
WHERE ac.accountR.branchR.bID = BranchId  
AND ac.accountR.balance = BalanceMax  
/

Result:



## 4.4 D

Find employees who are supervised by a manager and have accounts in the bank, displaying the branch address that the employee works in and the branch address that the account is opened with.

SELECT ca.accountR.branchR.bAddress.street AS BranchStreet,  
ca.accountR.branchR.bAddress.city AS BranchCity,  
ca.accountR.branchR.bAddress.postal\_code AS BranchPostalCode,  
EmployeeStreet,  
EmployeeCity,  
EmployeePostalCode  
FROM customerAccount ca,  
(SELECT e.niNum AS EmployeeNiNum,  
e.employeeAddress.street AS EmployeeStreet,  
e.employeeAddress.city AS EmployeeCity,  
e.employeeAddress.postal\_code AS EmployeePostalCode  
FROM employeeTable e  
WHERE e.supervisorId.employeePosition = 'MANAGER')  
WHERE ca.customerR.niNum = EmployeeNiNum  
/

Result:

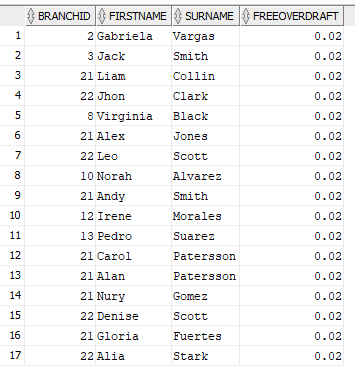


## 4.5 E

At each branch, find customers who have the highest free overdraft limit in all current accounts that are joint accounts, displaying the branch’s ID, the customer’s full names, the free overdraft limit in his/her current account.

SELECT BranchId,  
ac.customerR.firstName AS FirstName,  
ac.customerR.surName AS SurName,  
FODMax AS FreeOverDraft  
FROM (  
SELECT MAX(ca.accountR.limitOfFreeOD) AS FODMax,  
ca.accountR.branchR.bID AS BranchId  
FROM customerAccount ca  
WHERE ca.accountR.accType = 'CURRENT'  
GROUP BY ca.accountR.branchR.bID  
), customerAccount ac  
WHERE ac.accountR.branchR.bID = BranchId  
AND ac.accountR.limitOfFreeOD = FODMax  
/

Result:

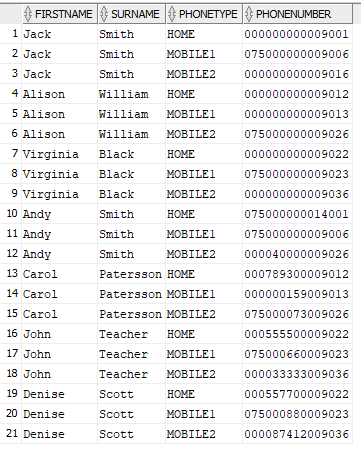


## 4.6 F

Find customers who have more than one mobile, and at least one of the numbers starts with 0750, displaying the customer’s full name and mobile numbers. COLLECTIONS must be used.

SELECT ct.firstName,  
ct.surName,  
tp.phoneType,  
tp.phoneNumber  
FROM customerTable ct,  
table(ct.customerPhone) tp  
WHERE customerId IN  
(  
SELECT c.customerId  
FROM customerTable c,  
table(c.customerPhone) ta,  
(  
SELECT COUNT(\*) AS NUMOFPHONES,  
p.customerId AS CustomerId2  
FROM customerTable p, table(p.customerPhone) t  
WHERE t.phoneType LIKE 'MOBILE%'  
GROUP BY p.customerid  
HAVING COUNT(\*) >= 2  
)  
WHERE ta.phoneNumber LIKE '%0750%'  
AND c.customerId = CustomerId2  
)  
/

Result:



## 4.7 G

Find the number of employees who are supervised by Mrs Smith, who is supervised by Mr Jones. REFERENCES must be used.

SELECT COUNT(\*)  
FROM employeeTable  
WHERE DEREF(supervisorId).surName = 'Smith'  
AND DEREF(DEREF(supervisorId).supervisorId).surName = 'Jones'  
/

Result:

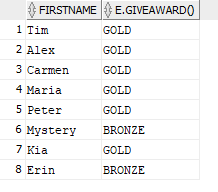


## 4.8 H

Award employees at the end of a year: gold medals for employees who have been working at the bank for more than 12 years and supervised more than 6 staff; silver medals for employees who have been working at the bank for more than 8 years and supervised more than 3 staff; bronze medals for employees who have been working at the bank for more than 4 years. Displaying winners’ names and Medal awarded (only displaying those who have been awarded). METHODS must be used.

ALTER TYPE employeeType  
ADD MEMBER FUNCTION giveAward  
RETURN VARCHAR2 CASCADE;  
/  
CREATE OR REPLACE TYPE BODY employeeType AS  
MEMBER FUNCTION giveAward RETURN VARCHAR2 IS  
sal VARCHAR2(10);  
BEGIN  
sal:= 'S/N';  
IF MONTHS\_BETWEEN(CURRENT\_DATE, joinDate) >= 144 THEN  
sal:= 'GOLD';  
END IF;  
IF MONTHS\_BETWEEN(CURRENT\_DATE, joinDate) < 144 AND MONTHS\_BETWEEN(CURRENT\_DATE, joinDate) >= 96 THEN  
sal:= 'SILVER';  
END IF;  
IF MONTHS\_BETWEEN(CURRENT\_DATE, joinDate) < 96 AND MONTHS\_BETWEEN(CURRENT\_DATE, joinDate) >= 48 THEN  
sal:= 'BRONZE';  
END IF;  
RETURN sal;  
END giveAward;  
END;  
/  
SELECT firstName, e.giveAward()  
FROM employeeTable e  
WHERE e.giveAward() != 'S/N'

Result:



# 5 Object-Relational vs Relational model

The principle of relational databases is based on the organization of information into small pieces, which are related to each other through the relationship of identifiers.

In a well designed relational database there is no duplicity of data.

It is a very robust system and is very stable and secure, until now it has been the most used system due to that. One of the reasons are the properties called ACID (atomicity, consistency, isolation, durability) which allows relational databases to be less vulnerable to errors.

Sometimes the data can be too complex for a relational database, so we can use an object relational database instead. Object relational databases gives flexibility as we don’t need to know before hand what type of data we need to store in it. We have the flexibility to be able to change the data stored from one subject to the next. It supports embedding objects and arrays within other objects and arrays, given great flexibility.

When dealing with huge amounts of data may be, we need to think if we can leave aside the benefits of the relational model and use instead an object relational database.

The reference ability allows us to get data without the need of joins, that is an advantage for the object relational model. On the other hand, one of the disadvantages could be considered the learning curve, as, specially at the beginning, it can be trickier to get the hand of it.

# 6 Deleting the database

There is a document submitted with this report called droppingTypesTables.sql where the following code is listed as well.

drop table "SYSTEM"."CUSTOMERACCOUNT" cascade constraints PURGE;

drop type "SYSTEM"."CUSTOMERACCOUNTTYPE" force;

drop table "SYSTEM"."ACCOUNTTABLE" cascade constraints PURGE;

drop type "SYSTEM"."ACCOUNTTYPE" force;

drop table "SYSTEM"."EMPLOYEETABLE" cascade constraints PURGE;

drop type "SYSTEM"."EMPLOYEETYPE" force;

drop table "SYSTEM"."CUSTOMERTABLE" cascade constraints PURGE;

drop type "SYSTEM"."CUSTOMERTYPE" force;

drop table "SYSTEM"."BRANCHTABLE" cascade constraints PURGE;

drop type "SYSTEM"."BRANCHTYPE" force;

drop type "SYSTEM"."PHONENESTED" force;

drop type "SYSTEM"."PEOPLETYPE" force;

drop type "SYSTEM"."ADDRESSTYPE" force;

drop type "SYSTEM"."PHONETYPE" force;