Object-Oriented Database Design

Coursework Submission for SET09107

Edinburgh Napier University 2017-2018 Trimester 2

David McCahon 40214392

February 2018

Contents

[Introduction 3](#_Toc508656746)

[Entity Relational Diagram 4](#_Toc508656747)

[Object-Relational Design 5](#_Toc508656748)

[SQL Statements 10](#_Toc508656749)

[Question A 10](#_Toc508656750)

[Question B 10](#_Toc508656751)

[Question D 11](#_Toc508656752)

[Question E 11](#_Toc508656753)

[Question F 12](#_Toc508656754)

[Question G 13](#_Toc508656755)

[Question H 13](#_Toc508656756)

[Relational Model vs Object-Relational Model 14](#_Toc508656757)

[Drop Statements 18](#_Toc508656758)

[References 19](#_Toc508656759)

# Introduction

This report will follow the re-design of a relational database into an object-oriented database. The initial database design is for a bank who operates several branches across the UK. Each branch has its own unique identifier and stores customers details.

Each customer opens an account which becomes associated with the branch they are opening it at with one or more of the following account types; Current Account and Savings Account which can also be joint-ownership of each account. Therefore, each account may have more than one customer and a customer can also open multiple accounts. Each branch employs employees into the following roles; Head, Manager, Project Leader, Accountant and Cashier. Each staff member has a supervisor at each branch apart from the Branch Manager (Head).

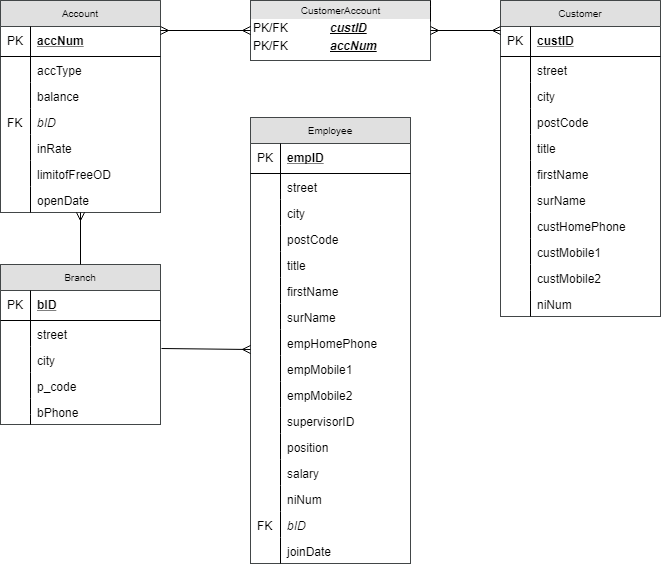
An employee award scheme will be intergrated into the database redesign where employees will be rewarded for their length of service to the bank and to supervisors who supervise a certain amount of staff.

This database re-design will provide the bank with greater reliability, flexibility and provide scalability and to a degree separation of concerns within their database, reduce maintenance and provide developers with high code reusability (Burelson, 2006) .

# Entity Relational Diagram

The initial entity relational model that the bank has currently in place is designed as below. This design will be redesigned to a new object-oriented database.

Figure 1: Entity Relational Diagram



Belongs to ⯈

⯇ Opens

⯇Works at

Associated With ⯆

# Object-Relational Design

The entity relational diagram will be redesigned to match a new object-oriented criteria however the semantics of the database must remain in place. Object Relational Databases follow the same rule as Object-oriented Programming in that instead of having the one table which is overloaded with repetitive, duplicated columns separate objects can be created and re-used.

Within the banks database design there are a lot of repetitive fields which can be avoided within the tables. To solve this problem, types are created where attributes are more related to each other than other attributes. This method prevents the repetition of fields which are constantly re-created within the entity relational model. Therefore, the types name\_type containing title, firstName and surName will be created as this combination is created within the employee and customer tables.

Furthermore, address\_type which is repetitively created within the branch, customer and employee tables will be created with the attributes street number, street, city and pCode. Although within the original Entity Relational Design, street number was not included within the data stored for employees or customers of the bank. However, Mr Jon Wright may live on Colinton Road which has hundreds if not over a thousand houses the chance of a letter for Mr Wright arriving at his house is very slim and therefore a street number will increase the accuracy of a letter getting delivered to Mr Wright.

The “Street Number” attribute will be stored as varchar2 to support customers or employees who;

1. Live in a house with a door number e.g. 1,104,134
2. Live in a communal flat e.g. 42G, 102A, 64F
3. Live in multi-floor flats e.g. – 42 1/1, 44 2/1, 89 2/3
4. Live in a house with a name e.g. – Burt Cottage, Hume Home, Alexander Villa

As seen within the branch, employee and customer tables phone numbers are redefined in each with the characteristics home phone, mobile phone 1 and mobile phone 2. However, due to recent technological developments of the mobile phone according to Statista, 2017 18% of the UK population do not have a landline/homephone. This would create a large amount of NULL values within the database. Moreover, the average mobile phone user would not own two mobile phones, and this would also create a lot of NULL values within the database. A more modern approach to fix this issue will be adapted based on the findings found.

Nested tables will be created which will allow for the flexibility of customers and employees who may have a landline and 2 mobile phones, a landline and mobile phone or just a landline and avoid mass amounts of null values within the database. Moreover, this method will support other phone numbers that will be linked to the employee or customer account included work phone number, next of kin phone number etc. Nested Tables are chosen over varrays due to nested tables can be queried more efficiently than varrays as they are treated like a table.

The database will be designed to ensure the integrity of the data entered into the database is of the same structure. Regex’s will be used to enforce strict data entry mechanisms which will enable the database to have the same structure throughout. The table below illustrates the table, column, accepted format and declined formats of potential data entered into the database.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tables** | **Columns** | **Regex** | **Accepted Format** | **Unaccepted Fomat** |
| employee\_table,  customer\_table | niNum | ^([A-Za-z]{2}\d{6}[A-Za-z])$ | PB234545D  QW112233X | P123456DF  P12B34BG |
| branch\_table customer\_table employee\_table | pcode | ^([A-PR-UWYZ0-9][A-HK-Y0-9][AEHMNPRTVXY0-9]?[ABEHMNPRVWXY0-9]? {1,2}[0-9][ABD-HJLN-UW-Z]{2}|GIR 0AA)$')) | DN3 6GB  SW42 4RG  GIR 0AA | SEW4 5TY  AA2C 4FG  AA2 4CV |

Table 1: Regex Constraints

By using regex to validate the data inserted into the database rogue information can be flagged and mistakes easily caught before customer information is unusable. National Insurance number follows a general guide of XX123456X, this keeps the format to the government defined standard and the same format throughout the database. Postcodes are more complicated with specific letters appearing and disallowed in certain places. These guidelines are set out by the UK Government (Government, 2012) who regulate postcodes within the UK. The regex implies the rules set out by the government and just not the general style of the postcode (Wade, 2017).

Customer\_table and employee\_table both share the same types of name\_type, address\_type and phone\_t. Therefore, person\_type will be created as a supertype which employee\_type and customer\_type will be created under and inherit the attributes from person\_type.

As niNum (National Insurance Number) is seen in both the customer and employee tables alongside the person\_type, it was initially apart of the design to include niNum within the person\_type. However, the restriction of whether the value to be NULL or not has left this design to be changed to adopt a more modern approach to banking. Within the employee\_table, the niNum should be NOT NULL as every employee who works at the branch must have the right to work in the UK and therefore a National Insurance Number is required. Whereas, customers can join the bank and create an account whether they are from the UK or international and therefore an international customer will not have a national insurance number. Furthermore, people under the age of 16 could create an account however are not assigned a national insurance number until the age of 16 and therefore niNum within the customer\_table will be set as NULL.

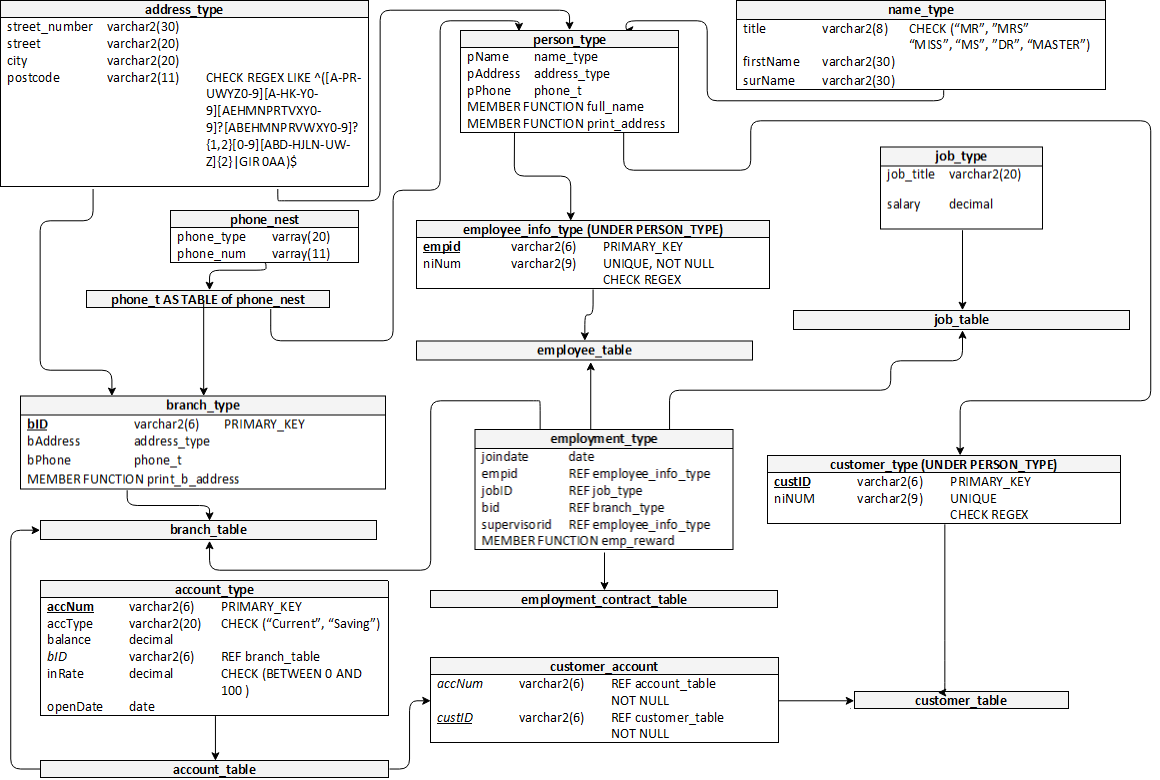


Figure 2: Object Relational Design

Within the database design employees are employed and work at a branch. An employee’s details are stored within the employee\_table which stores the employees name, address, phone numbers and national insurance number. It was initially a part of the design to include the entirety of the employment history within this table however, in a realistic scenario an employee requires a contract to be able to work. Therefore, employee\_contract\_table is created which inherits the data stored in the employee\_table (name, contact details and national insurance number) and the contract details of the employee – job title; salary; the branch that the employee works at; the supervisor of the employee and date the employee joined / when the contract started.

This design for employees enables the same employee record to be re-used in the instance that an employee changes their contract. For example, Jon Wright currently works at the Edinburgh Branch as an accountant however has been promoted to a manager. This allows Jon’s employee record to be re-used and a new contract is created for him stating his new manager job title, start date and his new updated manager. Another example could be Tiffany Cloud has taken maternity leave at the Glasgow branch. Within this employee\_contract\_table a record for Jon Wright can be created at the Glasgow branch to cover for Tiffany’s maternity leave. However, Jon’s original contract at the Edinburgh branch will remain intact and untouched. This design allows Jon to cover for Tiffany but does not affect his contract at the Edinburgh branch. To enhance this design, an employment\_type attribute could be added and store data such as “Perm, Temp, Contract or Leave”.

3 member functions will be created within the database to enable the ease of access to data which is frequently repeated throughout queries and to gain access to data required for employee rewards. Firstly, a Full Name member function will be created to allow for a person’s full name to be displayed. This is beneficially as instead of selecting an employees name by using the following 3 seen in Code1: Name Select.

Code 1: Name Select

**SELECT** e**.**empid**.**pname**.**surname**,**

e**.**empid**.**pname**.**firstname**,**

e**.**empid**.**pname**.**surname

**FROM** employment\_contract\_table e**;**

A simple full\_Name() memberfunction is used to concatenate the Title, Firstname and Surname attributes together. Moreover, this allows for the viewing of this data to be easily understood as “Mr”, “Jon”, “Wright” can now be viewed as “Mr Jon Wright”. The same style of member function is used in print\_address which takes the House Number, Street, City and Postcode and concatenates it together as seen in Code2: Name Member Function.

Code 2: Fullname Member Function

**SELECT** e.print\_name()

**FROM** employment\_contract\_table e**;**

The final memberfunction is designed to show medals awarded to employees for their length of service and the number of staff they supervise. This function firstly counts the number of times a supervisorid appears within the employment\_contract\_table. Secondly, the number of years the employee has worked with the branch is calculated by Equation 1: Years of Experience

Equation 1: Years of Experience

years\_of\_experience **:=** **((sysdate-**self**.**joindate**)/**365**);**

This function will calculate the employee’s experience to the date and therefore create a decimal value of the years worked. This will award employees with bronze if for example they have worked at the branch for 4 years and 1 day. Therefore, with the above two sub-functions, an If statement can reward each employee with the following medals;

1. If employee has more than 12 years of experience and supervised 6 staff, Gold Medal
2. If employee has more than 8 years’ experience and supervised 3 staff, Silver Medal
3. If employee has more than 4 years of experience, Bronze Medal
4. If employee has less than 4 years’ experience, No Medal Rewarded

Another design which was considered however was not implemented was to remove the job\_type as seen in figure2 and have variable job titles with different salary’s by manually assigning each employee their own salary. In the implemented design it is assumed that each person working in the same role all get paid the same. However, as reported by Jones, 2018, a BBC news reporter, men within the UK get paid 18.4% more than UK women. By having each job title assigned a salary which applies to all employees and not manually assigned to each employee allows for employees of the same job title to earn the same wage and generating a positive equal pay atmosphere regardless of employee’s gender, race, sexual orientation or disability.

Although a negative impact of the mentioned design above is that staff who are more experienced are paid the same as a member of staff who is perhaps not very good at their job or have entered the job with entry level requirements. This is a problem considering more experienced members of staff should be paid for the skills they have. To overcome this problem two potential methods could be implemented in future development of this database design;

1. Job titles are split into various levels which reflect an employee’s progression throughout the job with entry level being paid the base amount and more experience employees being paid relevant to their skillset. An example of this as discussed by HierarchyStructure, 2017 entry level accountants will be assigned the file clerk role earning the base level wage. However, as they progress throughout their career and gain experience can be promoted through the hierarchy to achieve a greater title and higher wage package. Moreover, this also fights the gender pay gap as regardless of gender, promotions should be awarded by experience.

1. Employees within the same job title can be paid the same base pay, however instead of receiving “gold, silver or bronze” medals they receive actual real-life awards which matter. Employees start of at the base salary and as they gain experience and duration of employment an attribute can be added to the employment table of a percentage bonus per year. For example, employees who earn “Gold Medal” earn a 15% bonus at the end of the year, “Silver Medal” 10% bonus and “Bronze Medal” 5% bonus. However, for this method to work more reasonings need to be considered to award employees medals compared to just how many staff supervised and duration of stay. This current method of awarding employees only awards employees at the top of the hierarchy – Management and Supervisors. Employees lower down in the hierarchy could be efficient, positive and make an impact on the company however can only ever receive a bronze award after 3 years.

# SQL Statements

## Question A

Display employees full name where their first name contains “on” and live in Edinburgh.

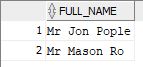
**SELECT**

e**.**print\_name**()** **AS** Full\_Name

**FROM** employee\_table e

**WHERE** e**.**pname**.**firstName **LIKE** '%on%' **AND** e**.**paddress**.**city **=** 'Edinburgh'**;**

Output:



## Question B

Find the number of savings account at each branch displaying the branch address and number of savings accounts.

**select**

a**.**BID**.**print\_b\_address**()** **AS** Address**,**

**COUNT(**a**.**acctype**)** **AS** Saving\_ACCOUNTS

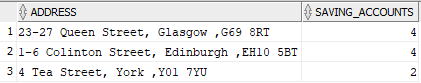
**FROM** account\_table a

**WHERE** acctype **=** 'Savings';

**GROUP BY** a**.**bid**,** a**.**bid**.**print\_b\_address**()**

**ORDER BY** Saving\_ACCOUNTS **DESC;**

Output:



## Question C

Find the customer with the highest savings account balance and their current accounts overdraft limit. Display full name, branchid, balance and overdraft limit.

**SELECT**

c**.**accnum**.**bid**.**bid **AS** Branch\_id**,**

c**.**custid**.**print\_name**()** **AS** Name**,**

c**.**accnum**.**balance **AS** Balance**,**

c2**.**accnum**.**limitoffreeod **AS** Overdraft

**FROM** **(**

**SELECT**

c**.**accnum**.**bid**.**bid **as** branch\_id**,**

c**.**accnum**.**acctype **AS** acctype**,**

**MAX(**c**.**accnum**.**balance**)** **AS** max\_balance

**FROM** customer\_account\_table c

**WHERE** c**.**accnum**.**acctype **=** 'Savings'

**GROUP** **BY** c**.**accnum**.**bid**.**bid**,** c**.**accnum**.**acctype**)** balance

**JOIN** CUSTOMER\_ACCOUNT\_TABLE c

**ON** c**.**accnum**.**bid**.**bid **=** balance**.**branch\_id

**AND** c**.**accnum**.**balance **=** balance**.**max\_balance

**AND** c**.**accnum**.**acctype **=** balance**.**acctype

**LEFT** **JOIN** customer\_account\_table c2

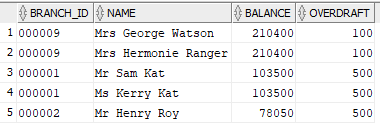
**ON** c2**.**custid**.**custid **=** c**.**custid**.**custid

**AND** c2**.**accnum**.**acctype **=** 'Current'

**ORDER BY** balance **DESC;**

**;**

Output:



## Question D

Find employees who are supervised by a manager and have accounts in the bank. Display branch address,

**SELECT**

e**.**empid**.**print\_name**()** **AS** FullName**,**

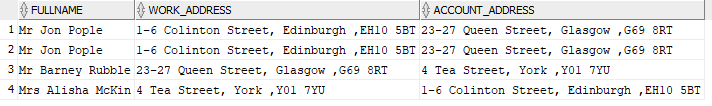
e**.**bid**.**print\_b\_address**()** **AS** Work\_Address**,**

c**.**accnum**.**bid**.**print\_b\_address**()** **AS** Account\_Address

**FROM** employment\_contract\_table e**,** customer\_account\_table c

**WHERE** e**.**supervisorid **IS** **NOT** **NULL** **AND** **(**e**.**empid**.**ninum **=** c**.**custid**.**ninum**);**

Output:



## Question E

Find the customers who have the highest overdraft limit at each branch that are joint accounts. Display branch id, customer full name and the free overdraft limit.

Output:

**SELECT**

c**.**accnum**.**bid**.**bid **AS** BRANCH\_ID**,**

c**.**custid**.**print\_name**()** **AS** Fullname**,**

c**.**accnum**.**limitofFreeOD **AS** Overdraft

**FROM (**

**SELECT**

c**.**accnum**.**bid **as** bid**,**c**.**accnum **as** accnum**,**

**COUNT(**c**.**custid**)** **AS** cust\_co

**FROM** customer\_account\_table c

**GROUP** **BY** c**.**accnum**.**bid**,**c**.**accnum**)**cust\_count**,** CUSTOMER\_ACCOUNT\_TABLE c

**WHERE** c**.**accnum**.**acctype **=** 'Current'

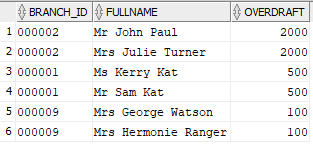
**AND** c**.**accnum **=** cust\_count**.**accnum

**AND** c**.**accnum**.**bid **=** cust\_count**.**bid

**AND** cust\_co **>** 1

**ORDER BY** Overdraft **DESC;**

**;**



## Question F

Find customers who have more than one mobile and at least one starts with 0770, display customers full name and mobile numbers.

**SELECT**

c**.**custid**,**

c**.**print\_name**(),**

t**.**phone\_num

**FROM**

**(**

**SELECT**

c**.**custid **AS** custid**,**

**count(**t**.**phone\_type**)** **AS** mob\_count**,**

phone\_type **AS** phonetype

**FROM** customer\_table c**,** **table(**c**.**pPhone**)** t

**WHERE** t**.**phone\_type **=** 'Mobile'

**GROUP** **BY** c**.**custid**,** phone\_type**)** phone\_nums**,** customer\_table c**,**

**table(**c**.**pPhone**)** t

**WHERE** c**.**custid **=** phone\_nums**.**custid

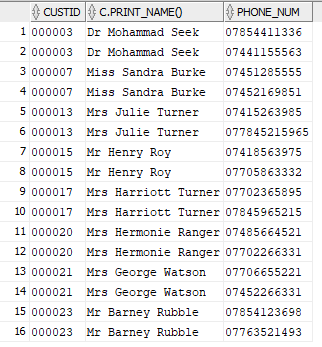
**AND** t**.**phone\_type **=** phone\_nums**.**phonetype

**AND** t**.**phone\_type **=** 'Mobile'

**AND** phone\_nums**.**mob\_count **>** 1

**ORDER** **BY** c**.**CUSTID**;**

Output:



Note: This shows customers with more than 1 mobile and only shows their mobile numbers. However, does not show numbers with 0770.

## Question G

Find number of employees supervised by Mrs Smith who is supervised by Mr Smith.

**SELECT**

e**.**empid**.**print\_name**()** **AS** Supervisor**,**

supervised**.**count\_super **AS** Staff\_Supervised

**FROM** **(**

**SELECT**

e**.**supervisorid**.**empid **AS** supid**,**

**COUNT(**e**.**supervisorid**)** **AS** count\_super

**FROM** employment\_contract\_table e

**WHERE** e**.**supervisorid **= (**

**SELECT**

e**.**empid **FROM** employment\_contract\_table e

**WHERE** e**.**empid**.**pname**.**title **=** 'Mrs'

**AND** e**.**empid**.**pname**.**surname **=** 'Jones' **AND**

e**.**supervisorid **=** **(**

**SELECT**

e**.**empid

**FROM** employment\_contract\_table e

**WHERE** e**.**empid**.**pname**.**title **=** 'Mr'

AND e**.**empid**.**pname**.**surname **=** 'Jones'**))**

**GROUP** **BY** e**.**supervisorid**.**empid **)**supervised**,**

employment\_contract\_table e

**WHERE** e**.**empid**.**empid **=** supervised**.**supid**;**

Output

https://i.gyazo.com/24df6d57657d1c968862d3ee1c9132e8.png

## Question 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 their names and Medal awarded (only displaying those who have been awarded).

**SELECT**

e**.**empid**.**print\_name**()** **AS** Full\_name**,**

e**.**emp\_reward**()** **AS** Medal

**FROM** employment\_contract\_table e

**WHERE** e**.**emp\_reward**()** **!=** 'No Medal Awarded'

**ORDER** **BY**

**CASE**

**WHEN** medal **=** 'Gold' **THEN** 0

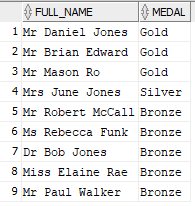
**WHEN** medal **=** 'Silver' **THEN** 1

**WHEN** medal **=** 'Bronze' **THEN** 2

**ELSE** 3

**END** **ASC;**

Output:



# Relational Model vs Object-Relational Model

Entity Relational Databases and Object Relational Databases are similar in the fact that relations are created between entities within the database usually related by primary keys and foreign keys. However, Object Relational Databases allow these relations to be extended into the data itself.

Within the Entity-Relational Model attributes are stored within an entity. These attributes are described as no relation and independent to each other and are simply populated with the data required. However, relations within the data can be created within the Object–Relational Model to allow users to map data together where data is more related to each other than another set of data. These relations are stored as types within the database. Tables inherit the structure of the associated type as well as types inherit from other types called within their structure. Inheritance allows for structures to be re-used throughout the database and minimise duplication.

For example, within the original database structure the employee table will be broken down to show relationships between the data (Figure 3: Relations).

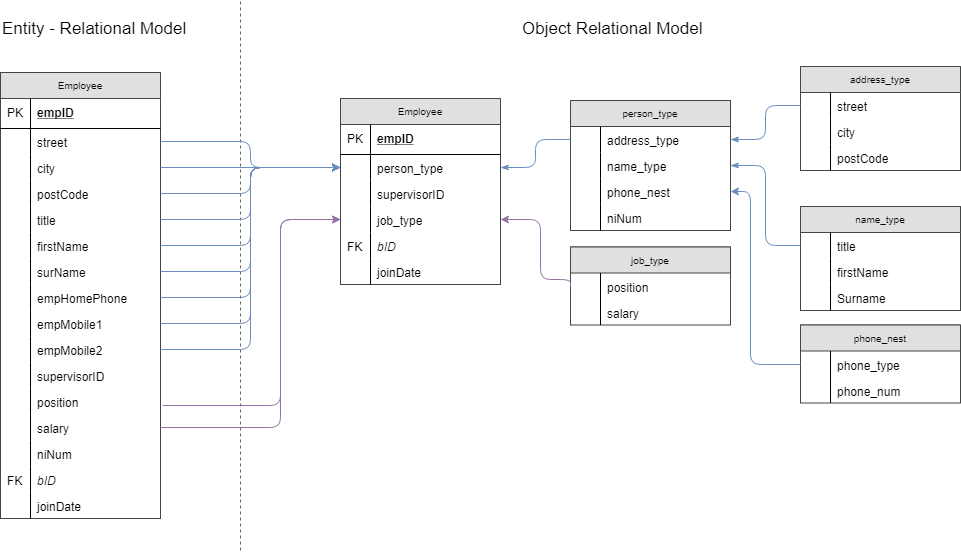


Figure 3: Relations

As seen above the attributes from the Entity – Relational model are split into types containing the other attributes they are related to. The above design shows 9 attributes from the Entity – Relational Model being condensed into one person\_type. This allows for the creation of tables to become easier and faster compared to the Entity-Relational model. Furthermore, a person\_type is created which contains information which is related to a person such as their name, address, phone numbers and national insurance numbers.

The design of implementing person\_type allows for the Object-Relational Model to improve on redundancy as it is used between multiple entities. For example, within the entity-relational design street, city, postcode, title, firstname, surname, homephone, mobile1 and mobile2 and all used within the employee table and also the customer table. Person type allows this data to be combined and inherited by each table it is used upon.

Within the object-oriented model methods are supported whereas within the entity relational model these are not. This is advantageous as basic or complex functions can be created and executed with ease. Examples of this include combining Title, Firstname and Surname together to print the fullname of a person within the person type. Firstly, this cuts down the quantity of selects required when writing queries. This also allows for ease of reading on the output of the query. Methods can be used in complex scenarios such as calculations and outputting a string within a series of if statements. For example, within my design a member function is created to award employees based on their experience and the number of staff they supervise (Code 3: Employee Reward).

**CREATE** **OR** **REPLACE** **TYPE** **BODY** employment\_type **AS**

MEMBER **FUNCTION** emp\_reward **RETURN** **VARCHAR2** **IS**

medal **varchar2(**20**);**

years\_of\_experience **INT;**

staff\_supervised **INT;**

**BEGIN**

**SELECT** **count(deref(**e**.**supervisorid**))**

**INTO** staff\_supervised

**FROM** employment\_contract\_table e

**WHERE** **deref(**e**.**supervisorid**).**empid **!=** **deref(**e**.**empid**).**empid**;**

years\_of\_experience **:=** **((sysdate-**self**.**joindate**)/**365**);**

**IF** years\_of\_experience **>** 12 **AND** staff\_supervised **>** 6 **then**

medal **:=** 'Gold'**;**

**ELSIF** years\_of\_experience **>** 8 **AND** staff\_supervised **>** 3 **then**

medal **:=** 'Silver'**;**

**ELSIF** years\_of\_experience **>** 4 **then**

medal **:=** 'Bronze'**;**

**ELSE**

medal **:=** 'No Medal Awarded'**;**

**END** **IF;**

**return** medal**;**

**END** emp\_reward**;**

**END;**

**/**

Code 3: Employee Reward

In the above example, employees are rewarded medals with the following criteria;

|  |  |
| --- | --- |
| Gold | Over 12 years experience and supervised more than 6 staff |
| Silver | Over 8 years of experience and supervise more than 3 staff |
| Bronze | Over 4 years of experience |

This member function prints the medal to the output for a query, in this case only medals awarded will be displayed (Figure 4: Medals Awarded).

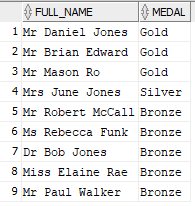


Figure 4: Medals Awarded

Another advantage of the Object Relational Model compared to the Entity Relational Model is the support of collections such as varrays and nested tables. Firstly, varrays allow for multiple sets of data to be stored on the same row. Secondly, nested tables allow for a table to be stored within the one row.

Within my design, Nested Tables are used to store the phone numbers of people and for the branch. Within the entity-relational model, if an employee did have not a phone number this would create multiple NULL values throughout the database. Whereas, within the Object-oriented model the nested table approach allows for numbers to be defined by a type “home” or “mobile”, this allows only true numbers to be listed and therefore will eliminate all null values.

This approach allows for a more simplistic view of the database, users can compare their own real-life scenarios to the database and understand its structure. However, implementing this design is more complex than implementing the Entity Relational Model due to its nested types and the complexity of inserting data into it.

For example, a comparison of the inserts can be viewed within Code 4: Entity Relational Insert and Code 5: Object Relational Insert

Code 5: Object Relational Insert

**INSERT** **INTO** employee **VALUES** **(**

'1'**,**

person\_type

**(**address\_type**(**'Oxford'**,**'London'**,**'LH12345'**),**

name\_type**(**'Mr'**,**'John'**,**'Smith'**),**

emp\_phone**(**phone\_nest**(**'home'**,**'01314567895'**),**phone\_nest**(**'mobile'**,**'07123456789'**),**phone\_nest**(**'mobile'**,**'079876543211'**))),**

'1'**,**

job\_type**(**'Head'**,**13200**),**

'1'**,**'13-JUN-2017'

**);**

Code 4: Entity Relational Insert

**INSERT** **INTO** employee **VALUES** **(**

'1'**,**'Oxford'**,**'London'**,**'LH12345'**,**'Mr'**,**'John'**,**'Smith'**,**'01314567895'**,**'07123456789'**,**'079876543211'**,**'1'**,**'Head'**,**13200**,**'PB11723C'**,**'1'**,**'13-JUN-2017

);

As seen above inserting into the Entity-Relational Model is simple as reading through the attributes within the entity. However, inserting into the Object-Relational Model looks more complex as each type which is used must be called upon within the insert statement. Although this is more complex compared to the Entity-Relational Model the insert within Code 2 can be easily read to determine what the data being inserted is related to. For example, as seen in Code 6: Address Insertion;

Code 4: Entity Relational Insert

The insertion can be easily read that the data stored belongs to an address.

Code 6: Address Insertion

**(**address\_type**(**'Oxford'**,**'London'**,**'LH12345'**),**

In conclusion, Object Relational Model is more complex in the terms of inserting into the data however can easily be read to the human eye.

# Drop Statements

----------DROPS----------

---TYPE DROPS---

**DROP** **TYPE** address\_type FORCE**;**

**DROP** **TYPE** phone\_nest FORCE**;**

**DROP** **TYPE** phone\_t FORCE**;**

**DROP** **TYPE** name\_type FORCE**;**

**DROP** **TYPE** person\_type FORCE**;**

**DROP** **TYPE** branch\_type FORCE**;**

**DROP** **TYPE** employee\_info\_type FORCE**;**

**DROP** **TYPE** job\_type FORCE**;**

**DROP** **TYPE** customer\_type FORCE**;**

**DROP** **TYPE** account\_type FORCE**;**

**DROP** **TYPE** employment\_type FORCE**;**

---TABLE DROPS---

**DROP** **TABLE** branch\_table**;**

**DROP** **TABLE** employee\_table**;**

**DROP** **TABLE** employment\_contract\_table**;**

**DROP** **TABLE** job\_table**;**

**DROP** **TABLE** customer\_table**;**

**DROP** **TABLE** account\_table**;**

**DROP** **TABLE** customer\_account\_table**;**

# References

Burelson, D. (2006) ‘Advantages and Disadvantages of HPLC’, in *HPLC*, pp. 1–13. doi: 10.1002/9780470079096.ch1.

Government (2012) ‘APPENDIX C Valid Postcode Format Guidance on recording valid postcodes’. Available at: https://www.gov.uk/government/uploads/system/uploads/attachment\_data/file/283357/ILRSpecification2013\_14Appendix\_C\_Dec2012\_v1.pdf (Accessed: 13 February 2018).

HierarchyStructure (2017) *Hierarchy of Accounting Jobs titles | Hierarchy Structure*. Available at: https://www.hierarchystructure.com/hierarchy-of-accounting-jobs/ (Accessed: 13 February 2018).

Jones, L. (2018) *Gender pay gap: What you need to know - BBC News*. Available at: http://www.bbc.co.uk/news/business-42918951 (Accessed: 13 February 2018).

Smartsheet (2018) *All about Relational Database Models | Smartsheet*. Available at: https://www.smartsheet.com/relational-database-modeling (Accessed: 26 February 2018).

Statista (2017) *• UK households with landline telephones Q1 2014-2017, by usage | Survey*. Available at: https://www.statista.com/statistics/386778/share-of-calls-enabled-landlines-in-uk-hoseholds/ (Accessed: 7 February 2018).

Wade, S. (2017) *Regular Expression Library*. Available at: http://regexlib.com/REDetails.aspx?regexp\_id=260&AspxAutoDetectCookieSupport=1 (Accessed: 13 February 2018).