Advanced Database Systems Report (SET09107)

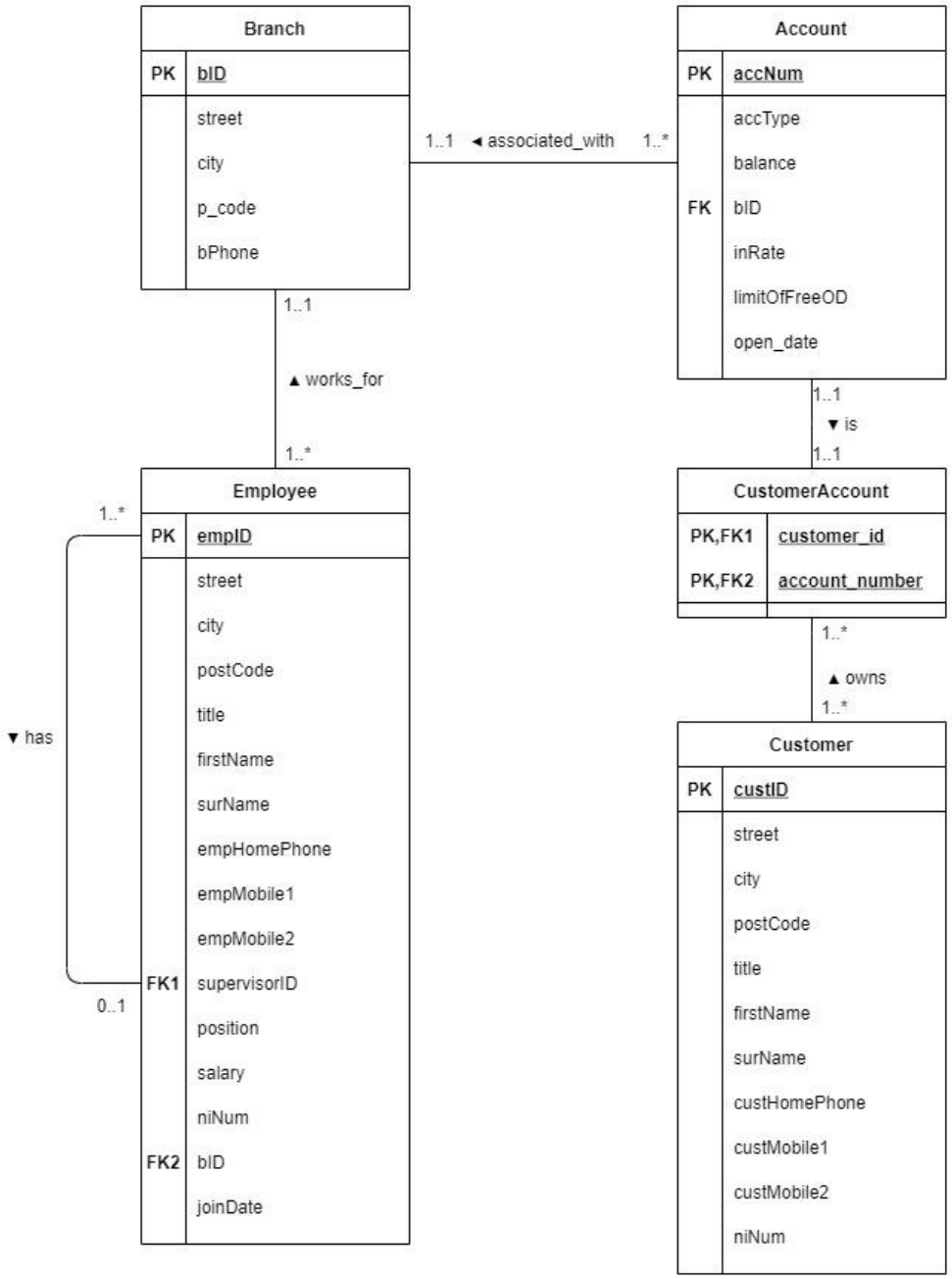
Valeri Vladimirov

40399682

School of Computing, Edinburgh Napier University, Edinburgh

40399682@napier.live.ac.uk

1 ER Diagram



This is the ER diagram corresponding to the relational database schema and the scenario. Each employee has exactly 1 supervisor and works for exactly 1 branch. However, if the employee has a job position as 'Head' he doesn't have a supervisor therefore the relationship is 0..1. A supervisor can supervise 1 or more employees. A branch can have 1 or many employees and 1 or many accounts can be created in the branch. Each account is associated with only 1 branch and has a 1 to 1 relationship with the Customer Account table. Each customer can have 1 or more customer accounts and each customer account can be a joint account (used by more than 1 customers).

2 Re-design

2.1 Structured Types

The new design of the database uses the object relational approach having 7 types, 3 subtypes and 6 tables. Using types and subtypes improves code redundancy as it can be seen from the Employee and Customer subtypes, which inherit properties from their Person supertype. The use of types in the object relation approach also reduces complexity, makes the code more reusable, readable and reliable, while also improving the quality and performance. Here are all of the structured types used:

Person Type – a structured type having attributes for fullname, address, ni_number and phone_numbers. This type is a supertype from which the Employee and Customer types inherit. The Person type was declared as NOT FINAL meaning that subtypes are allowed. The Person type has 3 methods: PrintName, PrintAddress, PrintPhoneNumbers. PrintName and PrintAddress are mainly used to make the printing of the properties faster and with less code in the SELECT statements. The PrintPhoneNumbers method was used to print all of the phone numbers of the person on one line (used for Question F).

```
/** Person Type **/
CREATE TYPE tp_Person AS OBJECT
](
    fullname tp_Name,
    address tp_Address,
    ni_number VARCHAR2(6),
    phone_numbers tp_Phone_Table
)NOT FINAL;
```

Name Type – a structured type having attributes for title, name and surname. It is a composite attribute which is used in other tables in the database. An example is the Person Type fullname attribute, which can be seen above.

```
/** Name Type **/
CREATE TYPE tp_Name AS OBJECT
(
    title VARCHAR2(5),
    firstname VARCHAR2(20),
    surname VARCHAR2(20)
)FINAL;
```

Address Type – a structured type having attributes for street, city, and postcode. This type is a supertype from which the Branch type inherits from. The Address type was declared as NOT FINAL meaning that subtypes are allowed. The Address type also has a method for PrintAddress, which reduces code in the SELECT statement (an example use is Question D).

```
/** Address Type **/
CREATE TYPE tp_Address AS OBJECT
(
    street VARCHAR2(20),
    city VARCHAR2(20),
    postcode VARCHAR2(8)
)NOT FINAL;
```

Phone Type – a structured type having attributes for phone_type and phone_number and also another type used to create a nested table for the customers, employees or branches which will have more than 1 number. Why this has been done will be explained in the Collections section.

```
/** Phone Type **/
CREATE TYPE tp_Phone AS OBJECT
(
    phone_type VARCHAR2(20),
    phone_number VARCHAR2(20)
)FINAL;

/

/** Nested Table of Phone Type **/
CREATE TYPE tp_Phone_Table AS TABLE OF tp_Phone;
```

Job Type – a structured type having attributes for id, position and salary. It is also a composite attribute for which there will be references in the Employee type. Declared as FINAL since there won't be any subtypes of it.

```
/** Job Type **/
CREATE TYPE tp_Job AS OBJECT
(
    id VARCHAR(6),
    position VARCHAR2(20),
    salary INTEGER
)FINAL;
```

Branch Type – a structured type having attributes for id and phone_numbers. It is a subtype of the Address supertype inheriting its properties and methods. Declared as FINAL since there won't be any subtypes of it.

```
/** Branch Subtype **/
CREATE TYPE tp_Branch UNDER tp_Address
(
    id VARCHAR2(6),
    phone_numbers tp_Phone_Table
)
    FINAL;
```

Account Type – a structured type having attributes for account_number, account_type, balance, interest_rate, branch_id, free_overdraft_limit and open_date. It includes a reference to the Branch type and is declared as FINAL since there won't be any subtypes of it.

```
/** Account Type **/
CREATE TYPE tp_Account AS OBJECT
(
    account_number INTEGER,
    account_type VARCHAR2(8),
    balance INTEGER,
    interest_rate VARCHAR2(8),
    branch_id REF tp_Branch,
    free_overdraft_limit INTEGER,
    open_date DATE
)
    FINAL;
```

Customer Type – a structured type having attributes for id. It is a subtype of the Person Type, which means that it inherits all of its properties and methods. Declared as FINAL since there won't be any subtypes of it.

```
/** Customer Subtype **/
CREATE TYPE tp_Customer UNDER tp_Person
(
    id VARCHAR2(6)
)
    FINAL;
```

Employee Type – a structured type having attributes for id, job_position, branch_id, join date, supervisor_id. It is a subtype of the Person Type, which means that it inherits all of its properties and methods. It also has an Award method, which is used for Question H and it counts how many years they have been working and how many employees they have supervised and then awards them based on the results. Declared as FINAL since there won't be any subtypes of it. It also includes various references to the Job, Branch types and also a reference was used for the supervisor (referencing to the same type).

```
/** Employee Subtype **/
CREATE TYPE tp_Employee UNDER tp_Person
(
    id VARCHAR2(6),
    job_position REF tp_Job,
    branch_id REF tp_Branch,
    join_date DATE,
    supervisor_id REF tp_Employee
)
    FINAL;
```

I have decided to create 6 tables to store the data from the structured types. When creating these tables I have applied various constraints to the attributes which can be seen in the Constraints section. The tables created are: Customer Table, Job Table, Branch Table, Employee Table, Account Table, Customer Account Table and can be seen in the DBCreating.sql file.

2.2 Inheritance

In the redesign of the database inheritance was used. This is due to the Object Relational approach, which is similar to Object Orientated Programming. Inheritance allows sub-types to inherit attributes from object types (supertypes) and this ensures reusability, reliability and code redundancy. An example of inheritance is how the Employee and Customer subtypes inherit attributes and methods from the Person Type.

```
CREATE TYPE tp_Customer UNDER tp_Person
```

2.3 References

References are features of the object relational model which behave as foreign keys. They are used as logical pointers to a row object and one of their main advantages is that they replace JOINs in queries. There are many examples of REF used in the redesign of the database, such as referencing the Job type, Branch type. Reference has also been used for the supervisor (in Employee Type), which makes all of the information about the Employee's supervisor available and quarriable. References are useful because they optimize the speed of queries and allow tables to contain more appropriate information.

```
job_position REF tp_Job,
branch_id REF tp_Branch,
join_date DATE,
supervisor_id REF tp_Employee
```

The "SCOPE IS" functionality can be used to restrict the REF to actual object tables as it can be seen when creating the Customer Account table.

```
CREATE TABLE tb_Customer_Account
(
    id REF tp_Customer SCOPE IS tb_Customer,
    account_number REF tp_Account SCOPE IS tb_Account
);
```

2.4 Methods

Member functions were used because they improve code redundancy (reduce repeated code) in the SELECT statements. They also make the code easier to understand. The methods are PrintName, PrintAddress, PrintPhoneNumbers and Award. More information about them was written above.

2.5 Constraints

Throughout the design I have used all of the constraints covered in the lectures. They were used to specify rules for the data in the tables, which ensure reliability and accuracy. They also improve the quality, performance and data integrity of the data.

Primary Key Constraint – was used to identify the primary key for the table, ensuring that these columns are unique.

```
CONSTRAINT id_const PRIMARY KEY (id),
```

"Check" Constraint – was used to validate incoming columns at row insert time, ensuring they match a given criteria. This constraint was used with the Not Null Constraint and for the account type checking if it is Current or Savings.

```
CONSTRAINT fullname_const_employee CHECK(fullname IS NOT NULL),
CONSTRAINT account_type_const CHECK(account_type IN ('Current', 'Savings')),
```

Not Null Constraint – as mentioned above it was used with the Check and it is used to specify that a column may never contain a NULL value.

```
CONSTRAINT job_position_const CHECK (job_position IS NOT NULL),
```

Unique Constraint – was used to ensure that there are no duplicate entries for all the column values in the table.

```
CONSTRAINT ni_number_const UNIQUE (ni_number),
```

2.6 Collections

Nested tables are one of the two collection types which Oracle supports. Nested tables (tables within tables) have been used for the current redesign of the database. They were used for the phone type, because there will be circumstances in which an employer,

customer or a branch can have more than one phone number. Nested tables provide various advantages such as having no limit (no upper bound) for how many phone numbers there are, and they increase the flexibility. Nested tables are unordered lists, and they prohibit NULL values when there aren't any phone numbers for the column.

```
/** Nested Table of Phone Type **/
CREATE TYPE tp_Phone_Table AS TABLE OF tp_Phone;

    CONSTRAINT phones_check_branch CHECK(phone_numbers IS NOT NULL)
)   NESTED TABLE phone_numbers STORE AS branch_phones_table;
```

2.7 Alternative Design

One alternative design would have been using Varrays rather than nested tables for the phone numbers. However, varrays were not implemented because they require a maximum size of the array to be specified which was considered a limitation for the current design and reduced the flexibility. The advantages of nested tables over varrays were that they have no limitation for the number of phone numbers entered (no upper bound) and that they are unordered lists (varrays are ordered lists), also they can store different data types. This makes the nested table more flexible and preferable for the current design.

Firstly, I thought of using Employee and Customer as types and not subtypes. However, the final design is with Employee and Customer being subtypes of the Person supertype, this decision was made because of the hierarchical approach and because it improves code redundancy. Also, inheritance had to be used for the object relational approach, and these subtypes were perfect examples.

3 SQL Statements

a) The first question uses the PrintName() method and searches for first names that have the 'ST' string, and the city is Glasgow. Examples show that it finds occasions where 'ST' is in the start of the name and in the middle.

```
/**
Question A
Find employees whose first name includes the string "st" and live in
Glasgow, displaying their full names.
**/

SELECT  e.PrintName() AS "FullName",
    e.address.city AS "City"
FROM    tb_Employee e
WHERE   UPPER(e.fullname.firstname) LIKE '%ST%'
AND     e.address.city = 'Glasgow';
```

	FullName	City
1	Mr. Stoyan Hibbert	Glasgow
2	Mr. Stanimir Vasilev	Glasgow
3	Mr. Stephan Perry	Glasgow
4	Mr. Tistan Jones	Glasgow

b) The second question uses the PrintAddress method to display the branch addresses. It searches for 'Savings' accounts in each branch and counts them. Grouping them and then ordering by descending. All 20 branches were included as examples.

```
/**
Question B
Find the number of saving accounts at each branch, displaying the
number and the branch's address.
**/

SELECT
    a.branch_id.id AS "Branch",
    a.branch_id.PrintAddress() AS "Address",
    count(a.account_type) AS "Accounts Count"
FROM
    tb_Account a
WHERE
    account_type LIKE 'Savings'
GROUP BY
    a.account_type, a.branch_id.PrintAddress(),  a.branch_id.id
ORDER BY
    count(a.account_type) DESC;
```

	Branch	Address	Accounts Count
1	E2	Firpark Street, Glasgow, G31 2AA	4
2	E4	Buckley Court, London, SE1 3FQ	3
3	C2	Rosemount Close, Glasgow, G21 2FF	3
4	D4	Phillipp Street, London, N1 5PE	3
5	B4	43 Chiltern Street, London, W1U 6LU	3
6	A2	George Square, Glasgow, G2 1DS	2
7	C1	The Quilts, Edinburgh, EH6 5RP	2
8	E3	Old Trafford, Manchester, M16 9LT	2
9	A3	Quenby Street, Manchester, M15 4HW	2
10	A1	Gorgie St., Edinburgh, EH14 1BF	2
11	E1	Meadowfield Avenue, Edinburgh, EH8 7NW	2
12	D3	7 Medway Close, Manchester, M5 5LD	2
13	D1	Kinellan Road, Edinburgh, EH12 6ES	2
14	A4	Berkeley Square, London, W1J 6EN	2
15	B1	Dublin Street, Edinburgh, EH3 6NT	2
16	C4	Chester Cottages, London, SW1W 8HG	2
17	D2	123 Fotheringay Road, Glasgow, G41 4LG	2
18	B2	19 Glamis Road, Glasgow, G31 4BJ	2
19	C3	Lockett Gardens, Manchester, M3 6BJ	2
20	B3	Saltergate Mews, Manchester, M5 4AD	1

c) The third question uses both PrintAddress and PrintName methods. It has subquery (nested SELECT in a SELECT statement), which is used to find the highest balance for each 'Savings' account. There are examples for each branch (20).

```sql
/**
Question C
At each branch, find customers who have the highest balance in their
savings account, displaying the branch address, their names, and the
balance.
**/

SELECT
    cl.account_number.branch_id.PrintAddress() AS "Branch Address",
    cl.id.PrintName() AS "Name",
    cl.account_number.balance AS "Balance"
FROM (
    SELECT
        c.account_number.branch_id.id AS branch,
        c.account_number.account_type AS account_type,
        MAX (c.account_number.balance) AS highest
        FROM
            tb_Customer_Account c
        WHERE
            c.account_number.account_type LIKE 'Savings'
        GROUP BY c.account_number.branch_id.id, c.account_number.account_type
    ) b
JOIN tb_Customer_Account cl
ON cl.account_number.account_type LIKE b.account_type
AND cl.account_number.balance LIKE b.highest
AND cl.account_number.branch_id.id LIKE b.branch
LEFT JOIN tb_Customer_Account c2
ON c2.id.id = cl.id.id
AND c2.account_number.account_type LIKE 'Basic'
ORDER BY cl.account_number.balance DESC;
```

	Branch Address	Name	Balance
1	19 Glamis Road, Glasgow, G31 4BJ	Mrs. Susan Clarkson	7424
2	Rosemount Close, Glasgow, G21 2FF	Mr. Kris Stephens	6325
3	Buckley Court, London, SE1 3FQ	Mr. Brett Moors	5670
4	Gorgie St., Edinburgh, EH14 1BF	Ms. Lorraine Earls	5367
5	Berkeley Square, London, W1J 6EN	Ms. Karissa Quincy	4654
6	Quenby Street, Manchester, M15 4HW	Mr. Earnest Harlan	4644
7	Meadowfield Avenue, Edinburgh, EH8 7NW	Mr. Moreen Winslow	4500
8	Phillipp Street, London, N1 5PE	Mr. Lally Stainthorpe	4300
9	43 Chiltern Street, London, W1U 6LU	Mrs. Philippa Tollemache	4213
10	Firpark Street, Glasgow, G31 2AA	Mrs. Zena Wade	3760
11	Lockett Gardens, Manchester, M3 6BJ	Mrs. Francis Blakesley	3726
12	George Square, Glasgow, G2 1DS	Mr. Prudence Palmer	3445
13	Saltergate Mews, Manchester, M5 4AD	Mrs. Zella George	2965
14	123 Fotheringay Road, Glasgow, G41 4LG	Mr. Rollo Banks	2800
15	Chester Cottages, London, SW1W 8HG	Mr. Ozzie Acker	2785
16	The Quilts, Edinburgh, EH6 5RP	Ms. Caiden Thorpe	2576
17	Kinellan Road, Edinburgh, EH12 6ES	Mr. Roxie Jacobson	2360
18	Dublin Street, Edinburgh, EH3 6NT	Mr. Pace Horn	1236
19	Old Trafford, Manchester, M16 9LT	Mr. Kelvin Younge	1200
20	7 Medway Close, Manchester, M5 5LD	Mrs. Poppy Walsh	870

d) The fourth question uses 3 methods: PrintName and both PrintAddress for branch and person. It finds if an employee works at a branch by checking if the names are equal and then checking if the supervisor is a 'Manager'.

```
/**
Question 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
    e.PrintName()    AS "Employee Name",
    c.account_number.account_number AS "Account Number",
    e.branch_id.PrintAddress() AS "Working Branch Address",
    c.account_number.branch_id.PrintAddress() AS "Account Branch Address"
FROM
    tb_Employee e, tb_Customer_Account c
WHERE
    c.id.PrintName() = e.PrintName()
AND
    e.supervisor_id.job_position.position LIKE 'Manager';
```

	Employee Name	Account Number	Working Branch Address	Account Branch Address
1	Mr. Rollo Banks	11111166	Meadowfield Avenue, Edinburgh, EH8 7NW	123 Fotheringay Road, Glasgow, G41 4LG
2	Mrs. Poppy Walsh	11111170	Saltergate Mews, Manchester, M5 4AD	7 Medway Close, Manchester, M5 5LD
3	Mr. Pace Horn	11111129	George Square, Glasgow, G2 1DS	Dublin Street, Edinburgh, EH3 6NT

e) The fifth question uses a few nested SELECT statements. The most inner select is used to find the account in the branch with the highest free overdraft limit. The second select is used to find if the account is a 'Current' one and also to check if it is a joint account by using the Count function. In the example there were 2 joint accounts in C3 branch one with 500 and one with 1000 OD limit.

```
/**
Question 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
    cl.account_number.branch_id.id AS "Branch ID",
    cl.account_number.account_number AS "Account Number",
    cl.id.PrintName() AS "FullName",
    cl.account_number.free_overdraft_limit as "Free OD Limit"
FROM (
    SELECT
    c.account_number.account_number AS AccNumber,
    COUNT(c.account_number.account_number)
    FROM (
        SELECT c.account_number.branch_id.id AS id,
            MAX(c.account_number.free_overdraft_limit) AS max
            FROM tb_Customer_Account c
            GROUP BY c.account_number.branch_id.id
        ) max, tb_Customer_Account c
    WHERE c.account_number.free_overdraft_limit = max.max AND
    c.account_number.branch_id.id = max.id
    AND c.account_number.account_type LIKE 'Current'
    HAVING COUNT(c.account_number.account_number) > 1
    GROUP BY c.account_number.account_number
    ) b
JOIN tb_Customer_Account cl
ON cl.account_number.account_number LIKE b.AccNumber;
```

	Branch ID	Account Number	FullName	Free OD Limit
1	C3	11111154	Mrs. Caelie Best	1000
2	C3	11111154	Mrs. Marigold Bullock	1000

f) The sixth question uses the PrintName method and the PrintPhoneNumbers method. The print phone numbers method is used to display all numbers in 1 string rather than showing only 1 of the customers numbers and adds comas. This query also has a subquery which checks for who has more than 1 'Mobile' numbers. Also checks if the number starts with '0760'.

```
/**
Question F
Find customers who have more than one mobile, and at least one of the
numbers starts with 0760, displaying the customer's full name and
mobile numbers. COLLECTIONS must be used.
**/

SELECT c.id AS "Customer ID",
c.PrintName() AS "Customer Name",
c.PrintPhoneNumbers() AS "Phone Numbers"
FROM tb_Customer c, TABLE(c.phone_numbers) t, (
                        SELECT c.id, COUNT(t.phone_type)
                        FROM tb_Customer c, table(c.phone_numbers) t
                        WHERE t.phone_type LIKE 'Mobile'
                        HAVING COUNT(t.phone_type) > 1
                        GROUP BY c.id
                    )  mobiles
WHERE c.id = mobiles.id
AND t.phone_number LIKE '0760%'
ORDER BY c.id;
```

	Customer ID	Customer Name	Phone Numbers
1	105	Ms. Caiden Thorpe	0760 9496 0041, 0131 9496 0042, 0131 9496 0043
2	107	Ms. Shawna Donalds	0760 9496 0035, 0131 9496 0036, 0131 9496 0037
3	114	Mrs. Zella George	0760 9496 0022, 0131 9496 0022, 0131 9496 0023
4	120	Mr. Kamryn Grover	0131 9496 1264, 0760 9496 6364, 0131 9236 0164

g) The seventh question counts the number of employees and uses References by checking which person is supervised by Mr Smith who is supervised by Mrs Jones.

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

SELECT COUNT(e.id) AS "Number of employees"
        FROM tb_Employee e
        WHERE e.supervisor_id.fullname.surname LIKE 'Smith'
        AND e.supervisor_id.supervisor_id.fullname.surname LIKE 'Jones'
        AND e.supervisor_id.fullname.title LIKE 'Mr'
        AND e.supervisor_id.supervisor_id.fullname.title LIKE 'Mrs';
```

	Number of employees
1	5

h) The last question uses the PrintName and Award methods. The Award method is used to count the number of people an employee has supervised and count the years the employee has worked for. Then using IF statements deciding which medal the employee deserves. It also has a WHERE clause which doesn't show the employees which don't have a medal awarded.

```
/**
Question H
Award employees at the end of a year: gold medals for employees who
have been working at the bank for more than 10 years and supervised
more than 8 staff; silver medals for employees who have been working
at the bank for more than 8 years and supervised more than 5 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.
**/

SELECT
        e.PrintName() AS "Employee Name",
        e.Award() AS "Employee Award"
        FROM tb_Employee e
        WHERE e.Award() != 'No Medal Awarded';
```

	Employee Name	Employee Award
1	Mr. Alexandur Petrov	Silver Medal
2	Mr. Stoyan Hibbert	Gold Medal
3	Mr. Nathan Adams	Bronze Medal
4	Mr. Alexandur Smith	Bronze Medal
5	Mr. Kurami Qnko	Bronze Medal
6	Ms. Violeta Racheva	Bronze Medal
7	Ms. Maikti Putkat	Bronze Medal

4 Advantages and disadvantages of the object-relational model

One of the disadvantages of the object-relational model is that some users can find it quite difficult and complicated since it utilizes the functionalities of both the relational data model and object orientated data model. Opposite to the object-relational model is the relational model, which is simple to learn, easier to use and more widely adopted.

One advantage of the object relational model over the relational model is that in the object relational model you can create and use methods to print the real values and not the types. Throughout the coursework I have used multiple methods, examples include the PrintName(), Award(), PrintAddress(). These methods allow the faster printing of the attributes (in one column rather than multiple) and also less code in the SELECT statement, which means it improves the speed of the queries and the performance. If the relational model had been used this wouldn't have been possible.

Another advantage of the object relational model over the relational one is the use of composite attributes. In the coursework an example of a composite attribute is the use of the Address(Street, City, Postcode) and Name(Title, Firstname, Surname) types in the Person type. The information is condensed into one column which saves time and memory, improving the queries performance.

The object relational model has the ability to map heterogeneous collections to tables, which the relational model cannot do. This makes possible to access and display different

data together, an example of that is the Phone numbers nested table, allowing to save and display multiple phone numbers together.

Encapsulation is another advantage of the object relational model over the relational model. Encapsulation improves the speed and performance of the queries by storing code inside of the database and this is not possible with the relational model.

Overall, the relational model has a few advantages such as being a simple model, supporting data independence, having good ad-hoc query facilities, good storage management, good concurrency and is fast and efficient. However, it has its drawbacks as mentioned above and also it can't run long duration transactions, express nested relationships, write methods, represent complex entities as a single unit, there is only a fixed number of types and can't sufficiently express data that does not map well to tables.

That is why the object relational model was chosen. Apart from the advantages mentioned above other advantages of the model are OOP features such as inheritance. This was used for the Person class being a supertype and the Employee, Customer types being subtypes, also for branch and address. Inheritance promotes reusability, reliability and reduces code redundancy. References is another feature which came in useful when doing the coursework. References to other tables were used because they improve the speed and performance for queries, making it quicker to call other types in the queries. Other advantages of the model include upwards compatibility with existing relational languages, preserving relational foundations, allowing attributes of tuple to have complex types.