Complex Databases in Relation to Banking

Oliver Thornewill von Essen
40210534@live.napier.ac.uk
Edinburgh Napier University
Advanced Database Systems (SET09107)

16 March 2018

Abstract

A bank has several branches in the United Kingdom, and the existing entity relational database must be redesigned and reconfigured into an Object Relational database. The limitations that come with entity relational diagram can be resolved with an Object Relational database. The current database has 5 tables. Branch, Account, Employee, Customer, CustomerAccount.

The current database stores information about the local branches that are identified by a **unique** branch code, an address (street, city, postcode) and a phone number.

Details about customer accounts are stored in the database. They are identified by a **unique** account number, account type (savings or current but not both), balance, interest rate, free overdraft limit. The date that the account was opened is recorded and every account is associated with exactly one branch.

Customer data is also recorded. Customers have an associated national insurance number, address(street, city, postcode), and their phone numbers. A customer may have multiple accounts with the bank, and an account may be owned by multiple customers as a joint account.

Similar to customers, employee data is also recorded. They also have an associated national insurance number, address (street, city, postcode), and their phone numbers. On top of these, an employee's salary, job position, their supervisor and a record for which branch the employee works for. The date on which the employee joined the branch is also recorded.

The database is redesigned using features from Oracle PL/SQL. Examples include an award function for outstanding employees or the use of types.

Keywords – Oliver, Thornewill, Von, Essen, Oracle PL/SQL, Advanced, Database, Systems, Napier, SET09107, 40210534

Contents

1 Entity Relational Diagram for Sample Data in Scenario 3

2 Database Redesign 4
- 2.1 Design of new database 4
- 2.2 Description of design 5
  - 2.2.1 Types 5
  - 2.2.2 References 5
  - 2.2.3 Constraints 5
  - 2.2.4 Nested tables (Collections) 5
  - 2.2.5 Member Functions 5
- 2.3 Alternative design 6
- 2.4 Flaws with the database 7

3 SQL queries including output 7
- 3.1 Find employees whose first name includes the sting 'on' and live in Edinburgh. . . 7
- 3.2 Find the number of saving accounts at each branch 7
- 3.3 At each branch, find customers who have the highest balance in their savings account. 8
- 3.4 Find employees who are supervised by a manager and have accounts in the bank. . 9
- 3.5 At each branch, find customers who have the highest free overdraft limit in all current accounts that are joint ccounts 9
- 3.6 Find customers who have more than one mobile, and at least one of the numbers starts with 0770 10
- 3.7 Find the number of employees who are supervised by Mrs Smith, who is supervised by Mr Jones. 11
- 3.8 Award employees at the end of a year 11

4 Discussion on Object Relational vs Entity Relational Models 12

5 Conclusion 13

6 Clear the Database 13

7 Abstract A 14

8 References 15

1 Entity Relational Diagram for Sample Data in Scenario

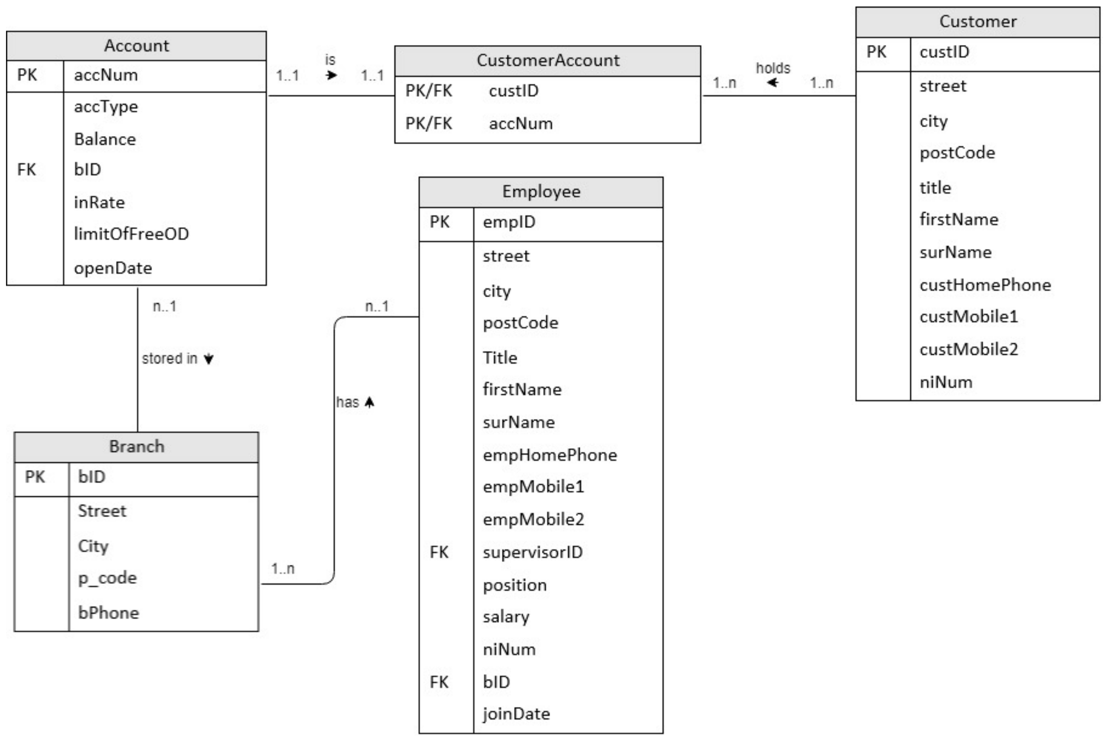

Figure 1: Entity Relational Diagram

2 Database Redesign

The purpose of redesigning this database is to to capture more of the semantics of object relational database features as extensively as possible without losing semantics of the previously implemented application. The name Object Relational Database stems from the idea that they have similar behavior to object oriented programming language features, for example, type inheritance.

2.1 Design of new database

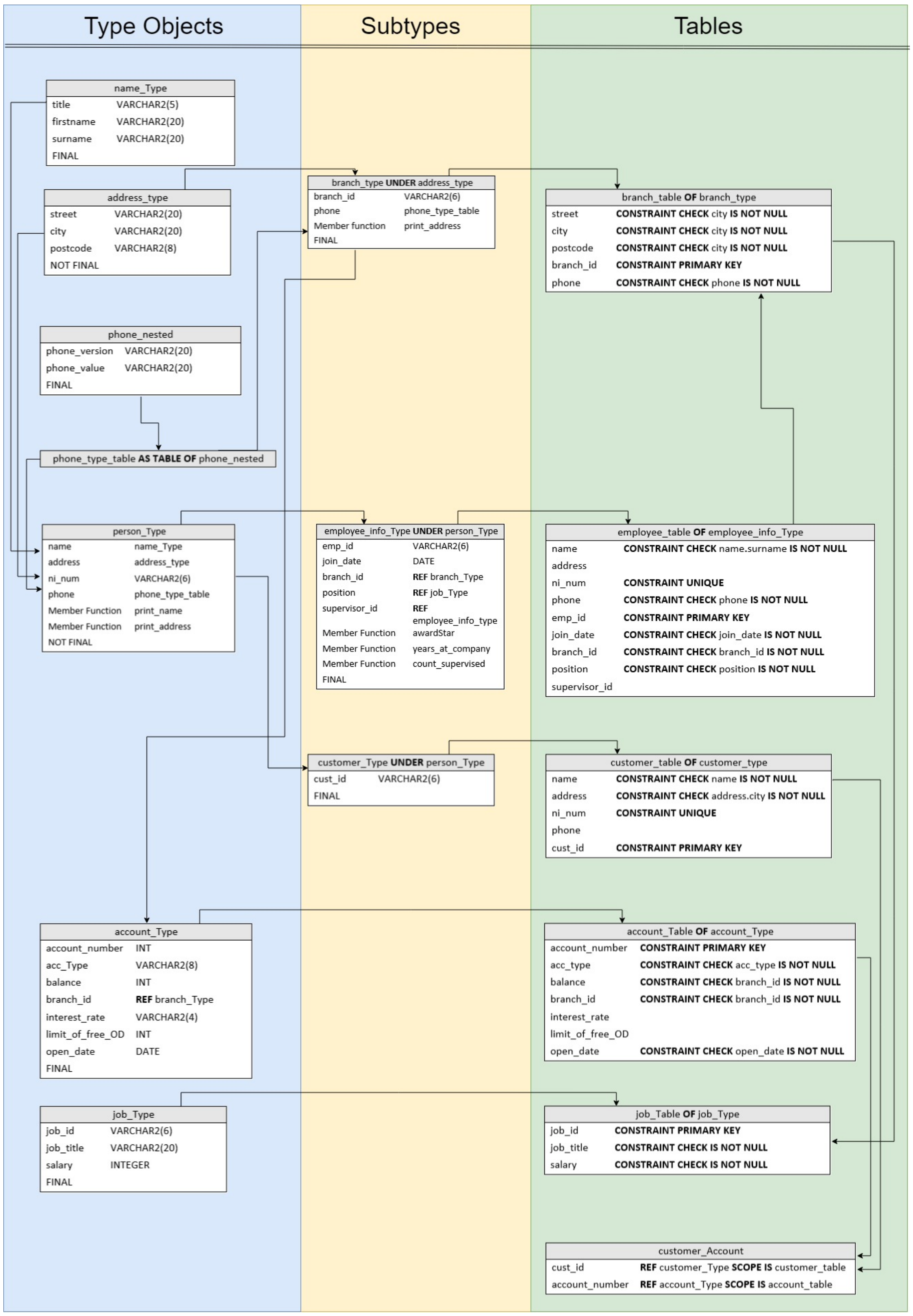
Type Objects
Subtypes
Tables
name_Type
title VARCHAR2(5)
firstname VARCHAR2(20)
surname VARCHAR2(20)
FINAL
address_type
street VARCHAR2(20)
city VARCHAR2(20)
postcode VARCHAR2(8)
NOT FINAL
phone_nested
phone_version VARCHAR2(20)
phone_value VARCHAR2(20)
FINAL
phone_type_table AS TABLE OF phone_nested
person_Type
name name_Type
address address_type
ni_num VARCHAR2(6)
phone phone_type_table
Member Function print_name
Member Function print_address
NOT FINAL
account_Type
account_number INT
acc_Type VARCHAR2(8)
balance INT
branch_id REF branch_Type
interest_rate VARCHAR2(4)
limit_of_free_OD INT
open_date DATE
FINAL
job_Type
job_id VARCHAR2(6)
job_title VARCHAR2(20)
salary INTEGER
FINAL
branch_type UNDER address_type
branch_id VARCHAR2(6)
phone phone_type_table
Member function print_address
FINAL
employee_info_Type UNDER person_Type
emp_id VARCHAR2(6)
join_date DATE
branch_id REF branch_Type
position REF job_Type
supervisor_id REF employee_info_type
Member Function awardStar
Member Function years_at_company
Member Function count_supervised
FINAL
customer_Type UNDER person_Type
cust_id VARCHAR2(6)
FINAL
branch_table OF branch_type
street CONSTRAINT CHECK city IS NOT NULL
city CONSTRAINT CHECK city IS NOT NULL
postcode CONSTRAINT CHECK city IS NOT NULL
branch_id CONSTRAINT PRIMARY KEY
phone CONSTRAINT CHECK phone IS NOT NULL
employee_table OF employee_info_Type
name CONSTRAINT CHECK name.surname IS NOT NULL
address
ni_num CONSTRAINT UNIQUE
phone CONSTRAINT CHECK phone IS NOT NULL
emp_id CONSTRAINT PRIMARY KEY
join_date CONSTRAINT CHECK join_date IS NOT NULL
branch_id CONSTRAINT CHECK branch_id IS NOT NULL
position CONSTRAINT CHECK position IS NOT NULL
supervisor_id
customer_table OF customer_type
name CONSTRAINT CHECK name IS NOT NULL
address CONSTRAINT CHECK address.city IS NOT NULL
ni_num CONSTRAINT UNIQUE
phone
cust_id CONSTRAINT PRIMARY KEY
account_Table OF account_Type
account_number CONSTRAINT PRIMARY KEY
acc_type CONSTRAINT CHECK acc_type IS NOT NULL
balance CONSTRAINT CHECK branch_id IS NOT NULL
branch_id CONSTRAINT CHECK branch_id IS NOT NULL
interest_rate
limit_of_free_OD
open_date CONSTRAINT CHECK open_date IS NOT NULL
job_Table OF job_Type
job_id CONSTRAINT PRIMARY KEY
job_title CONSTRAINT CHECK IS NOT NULL
salary CONSTRAINT CHECK IS NOT NULL
customer_Account
cust_id REF customer_Type SCOPE IS customer_table
account_number REF account_Type SCOPE IS account_table

Figure 2: Object Relational Diagram

2.2 Description of design

2.2.1 Types

Definition of Type: "A user-defined composite datatype that encapsulates a data structure along with the functions and procedures needed to manipulate the data" [1]. Object types are also known as super types. Variables forming the data structure are known as attributes. For example, the account_number inside the account_Type is an attibute. Object types reduce complexity by breaking down a large system into logical entities, creating software components that are modular, maintainable, and reusable [2].

Definition of Sub Type: "Sub types specify the same set of operations as their super type." Sub types do not act as super types, rather it places optional additional attributes in place [3]. Sub types can increase "reliability and improve readability, while also providing support for ANSI/ISO types.[3]

Application of types into this database: This design takes the object relational approach, using 10 types (7 object types and 3 sub types) which then create 6 separate tables with different properties. The use of types allows the tables to inherit properties from the type that it is derived from. Object Types can be used in multiple occasions which enables the prevention of copying and pasting, promoting more streamlined code. This can be seen with the person_Type as this has two sub types, inheriting the properties from the super type.

2.2.2 References

Referencing is a feature of the the object relational model and is similar to the use of foreign keys in the entity relational model. A reference is used when pointing to data from another table, and when querying for the data there is no JOIN function required [4]. An example of this in the created database design is in the employee_info_type with the "Supervisor" column. The clear advantage that references provide is that it takes all of the available data from the referenced table and inserts it into one column, keeping a condensed format, while also retaining all the data. In this case, the reference allows everything about the employee's supervisor to be queried. For example, it is therefore possible to access data regarding the supervisor's name, job title, salary, and all that is included within the type.

Application of references into this database: References have been sporadically placed throughout the database tables. This grants the database tables to contain more relevant data in a single table, eliminating the need for a JOIN function as required in an entity relational database during queries. This makes queries more optimized for speed.

2.2.3 Constraints

Constraints allow for better data management as they prohibit the insertion of fictitious data. It is possible to check fields are not NULL when inserting data, such as the Name field. In this database constraints were used in most fields, ensuring that the data rows inserted are not NULL values. Moreover constraints have been used in order to set the primary key where applicable.

2.2.4 Nested tables (Collections)

Nested tables are a tables **within** tables. The phone_Type in this database design is a nested table. This decision was made because there are occasions in which a branch/employee/customer only has one phone, but other occasions in which there are multiple phone numbers to store. Nested tables prevent NULL values in occasions where there are no phone numbers to fill the column. A further advantage of nested tables is that they condense multiple columns - of phone numbers - into a single column, without losing any data, and store these as unordered lists. In contrast to Varrays, nested tables do not have a specified maximum number of entries. Therefore, both nested tables and Varrays will violate the third normal form. However, this is preferred because while it will take more disk space to store everything. The combined affect of dropping storage pricing and increased speed makes the violation of the third normal form an acceptable tradeoff [5].

2.2.5 Member Functions

A special feature of Oracle PL/SQL is the accommodation of member functions. These are functions that handle and run calculations on the data of the type that the member function is featured inside. Three member functions have been defined for this database.

Firstly, this database allows an employer to run a function which will enable them to assign awards to outstanding employees - based on the number of employees an employee supervises, and on the number of years that the employee has been with the company. This member function is found inside the employee_info_type.

Secondly, there is a function to print an employee's full name, it is found inside the person_type. Normally this is not possible as the `select` statement used would provide the title, first name, and surname in separate columns. The member function overcomes this by selecting the data from both columns and through concatenation they can be output into the same column.

Table 1: Retrieving Full Name Without Function

Title	**First name**	**Surname**
Dr.	John	Smith
Mr.	Sam	Brown
Mrs.	Joanna	Tomson

The table shown above is taking three columns just to show an employee's full name, which is inefficient. The member function below allows the name to be presented into a singular column, while retaining all data.

Table 2: Retrieving Full Name With Member Function

Full Name
Dr. John Smith
Mr. Sam Brown
Mrs. Joanna Tomson

Finally, similar to the full name member function, there is a full address member function. This is found within both branch_type and person_type. It can be used to print the full address of both an employee or branch. The address type is used for both employee_table and branch_table. Inheritance enables the function to be enabled multiple times whereas the member function was only written once.

Table 3: Retrieving Address Without Member Function

Street	**City**	**Postcode**
2 Hermits Croft	Edinburgh	EH3 9FE
3 Bainfield Place	Edinburgh	EH11 1BF
10 Colinton Rd.	Edinburgh	EH10 5DT

The table shown above is taking three columns just to show an address, which is inefficient. The member function below allows the full address to be printed into a singular column, while retaining all data.

Table 4: My caption

Full Address
2 Hermits Croft, Edinburgh, EH3 9FE
3 Bainfield Place, Edinburgh, EH11 1BF
10 Colinton Rd., Edinburgh, EH10 5DT

2.3 Alternative design

An alternative design using **Varrays** instead of nested tables has been considered. Varrays store a fixed-size sequential collection of elements of the same type [6]. It was decided against Varrays because of the limitations including finite number of items, all being the same datatype. Nested tables have the clear advantage that they handle data as an **unordered** list, and that nested tables can store different data types.

2.4 Flaws with the database

The member function assigning awards to employees focuses on their job position. If an employee is a manager, they are going to supervise more people than a cashier or other lower level position (for example software developer) would. Based on the current method of assigning awards, lower position employees would never be selected for a medal. Perhaps the method for assigning awards should be based on productivity, assessing the number of tasks completed within the yearly quarter. To allow this change, the employee_table would have to be refactored, such that there are two new columns. The first column would be for the number of target tasks, and second column would be for indicating the number of completed target tasks. If the target tasks are 90% complete then the employee would get a ”Gold Award”.

The job_type sets a defined salary for each job type, meaning all managers get the same salary regardless of their experience. This problem could be overcome by assessing the employee’s productivity, again assessing the percentage of tasks completed relative to the employee’s targets. Instead of awarding a ”Gold Medal”, the award could be a bonus salary.

3 SQL queries including output

3.1 Find employees whose first name includes the sting ‘on’ and live in Edinburgh.

Listing 1: Query for 3a

```sql
SELECT  e.name.title || '. ' ||
        e.name.firstName || ' ' ||
        e.name.surname || ' is living in ' ||
        e.address.city
AS      "First name contains 'on'"
FROM    employee_table e
WHERE   INSTR(e.name.firstname,'on')>0
AND     e.address.city = 'Edinburgh';
```

Listing 2: Output for 3a

```
First name contains 'on'
------------------------------------------
Ms. Honour Tosteson is living in Edinburgh
Ms. Bonnie Davis is living in Edinburgh
Mr. Brandon Smith is living in Edinburgh
```

3.2 Find the number of saving accounts at each branch

Listing 3: Query for 3b

```sql
SELECT
    a.branch_id.branch_id AS "Branch ID",
    a.branch_id.city AS "City of branch",
    count(a.acc_type) AS "Number of savings accounts"
FROM
    account_table a
WHERE
    acc_type = 'Savings'
GROUP BY
    a.acc_type, a.branch_id.city,  a.branch_id.branch_id
ORDER BY
    a.branch_id.branch_id ASC;
```

Listing 4: Output for 3b

```
+-----------+-------------+-----------+
|           |             | Number of |
| Branch ID |    City     |  Savings  |
|           |             |  Accounts |
+-----------+-------------+-----------+
| C1        | Cambridge   | 3         |
+-----------+-------------+-----------+
| C2        | Bedford     | 3         |
+-----------+-------------+-----------+
| C3        | Northampton | 3         |
+-----------+-------------+-----------+
| C4        | Hitchin     | 3         |
+-----------+-------------+-----------+
| C5        | Stevenage   | 3         |
+-----------+-------------+-----------+
| C6        | Ware        | 3         |
+-----------+-------------+-----------+
| D1        | Barnet      | 3         |
+-----------+-------------+-----------+
| D2        | London      | 3         |
+-----------+-------------+-----------+
| D3        | London      | 3         |
+-----------+-------------+-----------+
| D4        | Romford     | 3         |
+-----------+-------------+-----------+
| D5        | Ipswich     | 3         |
+-----------+-------------+-----------+
| E1        | Edinburgh   | 4         |
+-----------+-------------+-----------+
| E2        | Edinburgh   | 3         |
+-----------+-------------+-----------+
| E3        | Edinburgh   | 3         |
+-----------+-------------+-----------+
| E4        | Edinburgh   | 3         |
+-----------+-------------+-----------+
| L1        | London      | 4         |
+-----------+-------------+-----------+
| L2        | London      | 3         |
+-----------+-------------+-----------+
| L3        | London      | 3         |
+-----------+-------------+-----------+
| L4        | London      | 3         |
+-----------+-------------+-----------+
| L5        | London      | 3         |
+-----------+-------------+-----------+
```

3.3 At each branch, find customers who have the highest balance in their savings account.

Listing 5: Query for 3c

```sql
SELECT
        c.account_number.branch_id.branch_id AS branch_ID,
        c.cust_id.cust_id AS cust_ID,
        c.cust_id.print_name() AS full_name,
        c.account_number.account_number AS accnum_of_savings,
        c.account_number.balance AS balance_of_savings
    FROM (
        SELECT
            c.account_number.branch_id.branch_id AS branch_id,
            c.account_number.acc_type AS acc_type,
            MAX (c.account_number.balance) AS max_balance
            FROM
                customer_account c
            WHERE
                c.account_number.acc_type = 'Savings'
            GROUP BY c.account_number.branch_id.branch_id, c.account_number.acc_type
        ) balance
    JOIN customer_account c
    ON
        c.account_number.branch_id.branch_id = balance.branch_id
    AND
        c.account_number.acc_type = balance.acc_type
    AND
        c.account_number.balance = balance.max_balance
    LEFT JOIN customer_account t2
    ON t2.cust_id.cust_id = c.cust_id.cust_ID
    AND t2.account_number.acc_type = 'Basic';
```

Listing 6: Output for 3c

```
+-----------+---------+-------------------+----------------+----------+
| Branch ID | Cust ID |    Customer Name  | Savings Account| Savings  |
|           |         |                   |     Number     |  Balance |
+-----------+---------+-------------------+----------------+----------+
| E4        | C1002   | Mr. Harold Jackson |       64758904 |    34634 |
+-----------+---------+-------------------+----------------+----------+
| L5        | C1006   | Mr. Earl Gray     |       92157579 |     9577 |
+-----------+---------+-------------------+----------------+----------+
| E1        | C1015   | Mr. Edward Scott  |        3621280 |     7587 |
+-----------+---------+-------------------+----------------+----------+
| E2        | C1020   | Mr. Michael Walker |       82516698 |    85932 |
+-----------+---------+-------------------+----------------+----------+
| E3        | C1024   | Mr. Bruce Barnes  |        4834745 |     9563 |
+-----------+---------+-------------------+----------------+----------+
| L1        | C1026   | Ms. Kathryn Rogers |       35953164 |     9636 |
+-----------+---------+-------------------+----------------+----------+
| L2        | C1033   | Ms. Jessica Young |        5545073 |     6853 |
+-----------+---------+-------------------+----------------+----------+
| L3        | C1039   | Ms. Marie Robinson |       95646311 |     8563 |
+-----------+---------+-------------------+----------------+----------+
```

3.4 Find employees who are supervised by a manager and have accounts in the bank.

Listing 7: Query for 3d

```
SELECT
    'Emp ID: ' || e.emp_id || ',  ' ||
    e.print_name()   AS employee,
    c.account_number.account_number AS "Account Number",
    e.supervisor_id.print_name() AS "Supervisor",
    e.supervisor_id.position.job_title AS "Supervisor Job Title"
FROM
    employee_table e, customer_account c
WHERE
    c.cust_id.name.firstname = e.name.firstname
AND
    c.cust_id.name.surname = e.name.surname
AND
    e.supervisor_id.position.job_title = 'Manager'
ORDER BY
    e.emp_id ASC;
```

Listing 8: Output for 3d

```
+-----------+---------------------+----------+----------------------+------------+
| Employee  |    Employee Name    | Account  |    Supervisor Name   | Supervisor |
|    ID     |                     | Number   |                      |    Job     |
|           |                     |          |                      |   Title    |
+-----------+---------------------+----------+----------------------+------------+
| 14        | Ms. Lois Adams      | 12667691 | Ms. Bonnie Davis     | Manager    |
+-----------+---------------------+----------+----------------------+------------+
| 19        | Ms. Kathryn Rogers  | 35953164 | Ms. Janet Brooks     | Manager    |
+-----------+---------------------+----------+----------------------+------------+
| 24        | Ms. Jessica Young   | 05545073 | Ms. Deborah Lee      | Manager    |
+-----------+---------------------+----------+----------------------+------------+
| 4         | Ms. Esther Moure    | 31866334 | Mr. Max Mustermann   | Manager    |
+-----------+---------------------+----------+----------------------+------------+
| 9         | Ms. Priscilla Moure | 47016078 | Ms. Bernice Kurkjian | Manager    |
+-----------+---------------------+----------+----------------------+------------+
```

3.5 At each branch, find customers who have the highest free overdraft limit in all current accounts that are joint ccounts

Listing 9: Query for 3e

```
SELECT
    c.account_number.branch_id.branch_id AS branch_id,
    c.cust_id.print_name() AS full_name,
    c.account_number.limit_of_free_OD AS free_od
    FROM (
        SELECT c.account_number.branch_id.branch_id AS branch_id,
            MAX(c.account_number.limit_of_free_OD) AS maxOD
            FROM customer_account c
            GROUP BY c.account_number.branch_id.branch_id
        ) maxOD, customer_account c
        WHERE c.account_number.limit_of_free_OD = maxOD.maxOD AND
         c.account_number.branch_id.branch_id = maxOD.branch_id
         ORDER BY c.account_number.branch_id.branch_id ASC;
```

Listing 10: Output for 3e

```
+-----------+--------------------+-----------+
|           |                    |   Free    |
| Branch ID |      Full Name     | Overdraft |
|           |                    |   Limit   |
+-----------+--------------------+-----------+
| E1        | Mr. Brandon Smith  |       200 |
+-----------+--------------------+-----------+
| E2        | Mr. Joshua Rogers  |       600 |
+-----------+--------------------+-----------+
| E3        | Ms. Katherine Jones |       500 |
+-----------+--------------------+-----------+
| E4        | Mr. Kevin Rivera   |       500 |
+-----------+--------------------+-----------+
| L1        | Ms. Judy Taylor    |       500 |
+-----------+--------------------+-----------+
| L2        | Ms. Amanda Parker  |       800 |
+-----------+--------------------+-----------+
| L3        | Ms. Marie Robinson |       800 |
+-----------+--------------------+-----------+
| L5        | Mr. Frank Long     |       500 |
+-----------+--------------------+-----------+
```

3.6 Find customers who have more than one mobile, and at least one of the numbers starts with 0770

Listing 11: Query for 3f

```
SELECT
    c.cust_id,
    c.print_name(),
    t.phone_value
FROM
    (SELECT c.cust_id AS cust_id, count(t.phone_version) AS mob_count, phone_version AS ↩
     phone_version
    FROM customer_table c, table(c.phone) t
    WHERE t.phone_version = 'Mobile'
    AND t.phone_value LIKE '07%'
    GROUP BY c.cust_id, phone_version) phone_nums, customer_table c, table(c.phone) t
WHERE c.cust_id = phone_nums.cust_id
AND t.phone_version = phone_nums.phone_version
AND t.phone_version = 'Mobile'
AND phone_nums.mob_count > 1
ORDER BY c.cust_id;
```

Listing 12: Output for 3f

```
+-------------+--------------------+---------------+
| Customer ID | Customer Full Name | Mobile Number |
+-------------+--------------------+---------------+
| C1003       | Mr. Arthur Cook    | 07800900791   |
+-------------+--------------------+---------------+
| C1003       | Mr. Arthur Cook    | 07700 900518  |
+-------------+--------------------+---------------+
| C1007       | Mr. Frank Long     | 07600900791   |
+-------------+--------------------+---------------+
| C1007       | Mr. Frank Long     | 07700 900125  |
+-------------+--------------------+---------------+
| C1013       | Mr. Brandon Smith  | 07700 900286  |
+-------------+--------------------+---------------+
| C1013       | Mr. Brandon Smith  | 07500900791   |
+-------------+--------------------+---------------+
| C1019       | Mr. Joshua Rogers  | 07700 900890  |
+-------------+--------------------+---------------+
| C1019       | Mr. Joshua Rogers  | 07400900791   |
+-------------+--------------------+---------------+
| C1026       | Ms. Kathryn Rogers | 07300900791   |
+-------------+--------------------+---------------+
| C1026       | Ms. Kathryn Rogers | 07700 900694  |
+-------------+--------------------+---------------+
| C1033       | Ms. Jessica Young  | 07700 900219  |
+-------------+--------------------+---------------+
| C1033       | Ms. Jessica Young  | 07700 900791  |
+-------------+--------------------+---------------+
| C1038       | Ms. Laura Young    | 07200900791   |
+-------------+--------------------+---------------+
| C1038       | Ms. Laura Young    | 07700 900041  |
+-------------+--------------------+---------------+
| C1041       | Mr. Jonathan Saxo  | 07100900791   |
+-------------+--------------------+---------------+
| C1041       | Mr. Jonathan Saxo  | 07700 900833  |
+-------------+--------------------+---------------+
```

3.7 Find the number of employees who are supervised by Mrs Smith, who is supervised by Mr Jones.

Listing 13: Query for 3g

```
SELECT
    COUNT(e.print_name()) as number_of_employees,
    e.supervisor_id.print_name() AS supervisor_of_employees,
    (SELECT e.supervisor_id.print_name() FROM employee_table e WHERE e.supervisor_id.name.Surname↩
      = 'Jones') AS supervisor_of_Mrs_Smith
    FROM employee_table e
    WHERE
        e.supervisor_id.name.firstname = 'Jourdan' AND
        e.supervisor_id.name.surname = 'Smith'
        AND e.supervisor_id.emp_id = (
            select e.emp_id FROM employee_table e
            WHERE
                e.supervisor_id.name.surname = 'Jones')
    GROUP BY e.supervisor_id.print_name() ;
```

Listing 14: Output for 3g

```
+-------------------------------+-------------------------+--------------------------+
| Count of supervised employees | Supervisor of Employees | Supervisor of Supervisor |
+-------------------------------+-------------------------+--------------------------+
| 3                             | Mrs. Jourdan Smith      | Mr. James Jones          |
+-------------------------------+-------------------------+--------------------------+
```

3.8 Award employees at the end of a year

Listing 15: Query for 3h

```
SELECT
    e.emp_id AS "Employee ID" ,
    e.print_name() AS "Employee Full Name" ,
    e.years_at_company() AS "Years at Company",
    e.count_supervised() || ' people' AS "Supervises",
    e.awardStar() AS "Awarded Medal"
FROM
    employee_table e
WHERE
    e.awardStar() != 'No Medal Awarded';
```

Listing 16: Output for 3h

```
+-------------+----------------------+---------+-------------+---------------+
| Employee ID | Employee Full Name   |  Years  |  Number of  | Awarded Medal |
|             |                      |   at    |  employees  |               |
|             |                      | company | supervising |               |
+-------------+----------------------+---------+-------------+---------------+
| 1           | Mr. John Smith       | 14      | 9 people    | Gold Medal    |
+-------------+----------------------+---------+-------------+---------------+
| 2           | Mr. Max Mustermann   | 9.5     | 4 people    | Silver Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 3           | Ms. Candace Gibbon   | 8.5     | 4 people    | Silver Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 4           | Ms. Esther Moure     | 9.5     | 1 people    | Bronze Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 7           | Ms. Bernice Kurkjian | 4.5     | 2 people    | Bronze Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 12          | Ms. Bonnie Davis     | 4.5     | 2 people    | Bronze Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 17          | Ms. Janet Brooks     | 4.5     | 2 people    | Bronze Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 22          | Ms. Deborah Lee      | 4.5     | 2 people    | Bronze Medal  |
+-------------+----------------------+---------+-------------+---------------+
| 41          | Mr. James Jones      | 4.5     | 1 people    | Bronze Medal  |
+-------------+----------------------+---------+-------------+---------------+
```

4 Discussion on Object Relational vs Entity Relational Models

Entity Relational: The entity relational model uses a collection of tables to represent both the data and relationships among the data. The entity relational model uses **relational keys** such as the Primary and Foreign key. The entity relational model has the advantage that it is simple which facilitates a less steep learning curve therefore making it more widely adopted in businesses. It has good ad-hoc query facilities and good storage management (for example data backup and recovery) [7]. Despite the ease of use, there are still problems with the entity relational model that the object relational model can do. entity relational model does **not** sufficiently express data which does not map well into tables (heterogeneous collections), define types with nested relationships, represent complex entities as a single unit (for example address), write methods, and finally run long duration transactions [7].

Object Relational: The object oriented model is an extension of the entity relational model with notions of encapsulations, methods, and object identity. It captures the semantics of objects supported in object oriented programming, including object oriented concepts [7]. Examples of these concepts include; complex objects; object identity; encapsulation; extensibility; class hierarchies and **inheritance**. The Object-**Relational** model refines the Object-**Oriented** model combining features of object oriented and entity relational models. the object relational model extends the relational data model through the use of object orientation and constructs to deal with added data types. It further allows attributes of tuples to have complex types, including non atomic values such as nested relations. The object relational model preserves relational foundations, in particular the declarative access to data, while extending modeling power. Problems with the object relational Approach are that it is more complicated, therefore less widely adopted than the entity relational model as it is more time consuming to learn and maintain it.

It was stated earlier that entity relational models do not map heterogeneous collections to tables well. The object relational approach has the ability to express these. A heterogeneous collection is where there are two different sets of data, however there are some features that are equal in these.

Set1 = (**Branch** = B1, **AccountNumber** = 15968547, **Type** = Current,
Branch = B1, **AccountNumber**= 85965854, **Type**= Current,
Branch = B1, **AccountNumber**= 75844269, **Type**= Savings)

Set2 = (**Branch** = B1, **AccountNumber** = 15968547, **Balance** = $322.04,
Branch = B1, **AccountNumber** = 85965854, **Balance** = $123.05,
Branch = B1, **AccountNumber** = 75844269, **Balance** = $90.85)

In this example, the data sets are heterogeneous, however the sets have certain common properties. The branch and AccountNumbers are identicle, however in Set1 it is possible to see the Type, and in Set2 it is possible to see the Balance. object relational databases make it possible to access and display all the data together [8].

The object relational model can - through the use of types - condense the number of total columns in a table. One type can contain many attributes, however these attribute values do not go into another column[7]. An example of this is that in a customer table where there are fields for the address (Street, City, and Postcode), they can be condensed into an address column without losing any of the information.

A key feature that the object relational model has that the entity relational model does not have is the ability to write and use **methods**. A method is defined by a "procedure or function that can operate on the attributes of a type" [9]. Methods are also known as member functions, and only operate on a type that they are included in. Methods require an extra type body which extends the regular type definition. A basic example of a method is a "Print Name" function. Regarding this database design, this can be placed inside the name_Type. The member body then calls the function to select the Title, firstname, and surname. This allows the full name to be printed into one column instead of multiple. There are much more complicated applications of member functions, including the member function that will award employee's based on their rating earlier in the report.

Encapsulation is a feature that the object relational model offers while the entity relational model does not. Encapsulation is a way of streamlining the performance of the database, making queries run quicker. Encapsulation provides many features in itself. Member functions can be stored inside the database, making it possible to cache the functions and allow them to execute very quickly. The code becomes independent because it can be encapsulated into stored procedures allowing functions to be executed by external programs. A further advantage of code being stored inside the database is that it eliminates the need to find SQL in external programs in the event that it needs to be changed, as all the SQL is in one place [10].

5 Conclusion

In summary, this report has discussed the differences between Object Relational databases against entity relational databases. The initial entity relational diagram has been refactored into a new design that includes semantics of the Object Relational database model. Examples of the Object Relational semantics include types, references, constraints, nested tables (collections), and member functions.

6 Clear the Database

The sequence of commands used to drop all the tables and types such that when run in sequential order everything will be removed.

Listing 17: Drop all types and tables

```sql
/***
Force drop types
***/

DROP TYPE name_type FORCE;
DROP TYPE address_type FORCE;
DROP TYPE branch_type FORCE;
DROP TYPE person_type FORCE;
DROP TYPE job_Type FORCE;
DROP TYPE employee_info_type FORCE;
DROP TYPE customer_type FORCE;
DROP TYPE account_type FORCE;
DROP TYPE phone_nested FORCE;
DROP TYPE phone_table FORCE;

/***
Drop tables.
***/

DROP TABLE job_table;
DROP TABLE employee_table;
DROP TABLE account_table;
DROP TABLE branch_table;
DROP TABLE customer_table;
DROP TABLE customer_account;
```

7 Abstract A

Table 1: Branch

Branch ID	Street	City	Postcode	Phone number
901	Market	Edinburgh	EH1 5AB	01311235560
908	Bridge	Glasgow	G18 1QQ	01413214556

Table 2: Account

Account Number	Account Type	Balance	Branch ID	Interest Rate	Free Overdraft	Open Date
1001	Current	820.50	901	0.005	800	01-May-11
1010	Savings	3122.20	901	0.02		08-Mar-10
8002	Current	200	908	0.005	100	05-May-09

Table 3: Employee

Employee ID	Street	City	Postcode	Title	First Name	Surname	Home Phone	Mobile Phone	Supervisor ID	Job Title	Salary	National Insurance Number	Branch ID	Join Date
101	Dart	Edinburgh	EH10 5TT	Mrs	Alison	Smith	01312125555	07705623443 07907812345		Head	50000	NI001	901	01-Feb-08
105	New	Edinburgh	EH2 4AB	Mr	John	William	01312031990	07902314551 07701234567	101	Manager	40000	NI010	901	04-Mar-09
108	Old	Edinburgh	EH9 4BB	Mr	Mark	Slack	01312102211		105	Accountant	30000	NI120	901	01-Feb-12
804	Adam	Edinburgh	EH1 6EA	Mr	Jack	Smith	01311112223	0781209890	801	Leader	35000	NI810	908	05-Feb-14

Table 4: Customer

Customer ID	Street	City	Postcode	Title	First Name	Surname	Home Phone	Mobile Phone	National Insurance Number
1002	Adam	Edinburgh	EH1 6EA	Mr	Jack	Smith	01311112223	0781209890 0771234567	NI810
1003	Adam	Edinburgh	EH1 6EA	Ms	Anna	Smith	01311112223	0770111222	NI010
1098	New	Edinburgh	EH2 8XN	Mr	Liam	Bain	01314425567		NI034

Table 5: Customer Account

Customer ID	Account Number
1002	1001
1002	1010
1003	1010
1098	8002

Figure 3: Sample Data for Task 1.

8 References

References

[1] Oracle, "Pl/sql user's guide and reference,"

[2] Oracle, "Object types,"

[3] Oracle, "Overview of pl/sql subtypes,"

[4] T. Peng, "Inheritance references and methods," Feb. 2018.

[5] B. Consulting, "Oracle data structure denormalization," July 2015.

[6] TutorialsPoint, "Pl/sql arrays,"

[7] T. Peng, "Object-relational databases," Feb. 2018.

[8] P. Buneman and A. Ohori, "Using kinds to represent heterogeneous collections in a static type system," Aug 1990.

[9] "5methods: Using pl/sql."

[10] D. Burleson, Apr 2016.