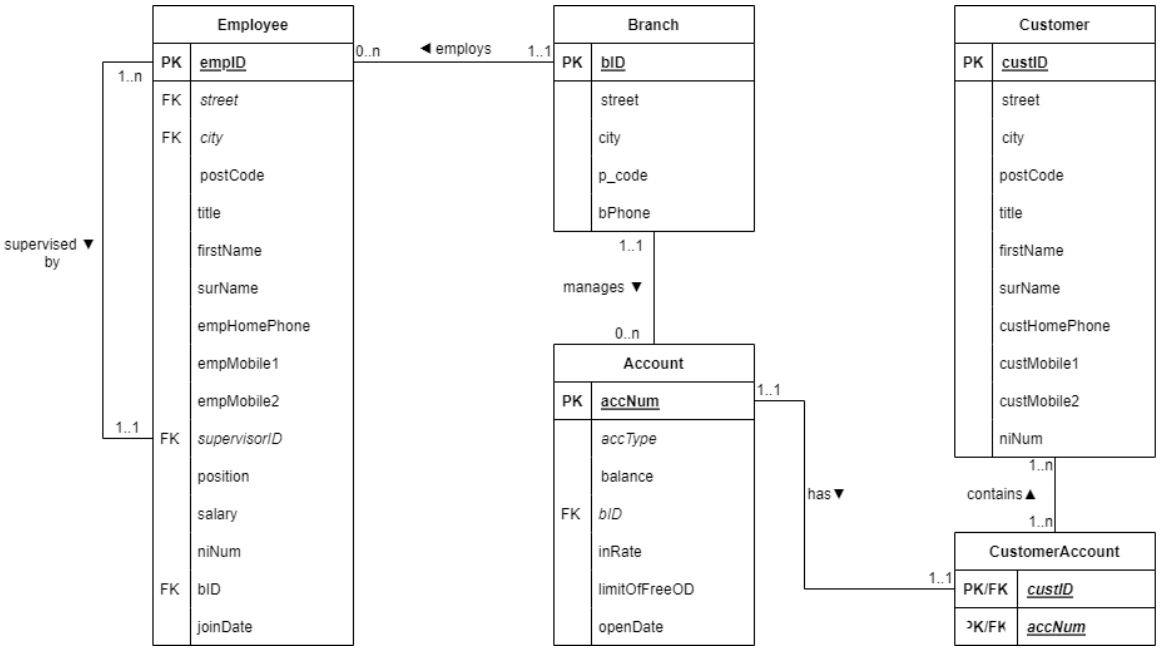
SET09107: Advanced Database Systems

2023/24 Coursework

Name: Michael Mackenzie

Matriculation Number: 40552802

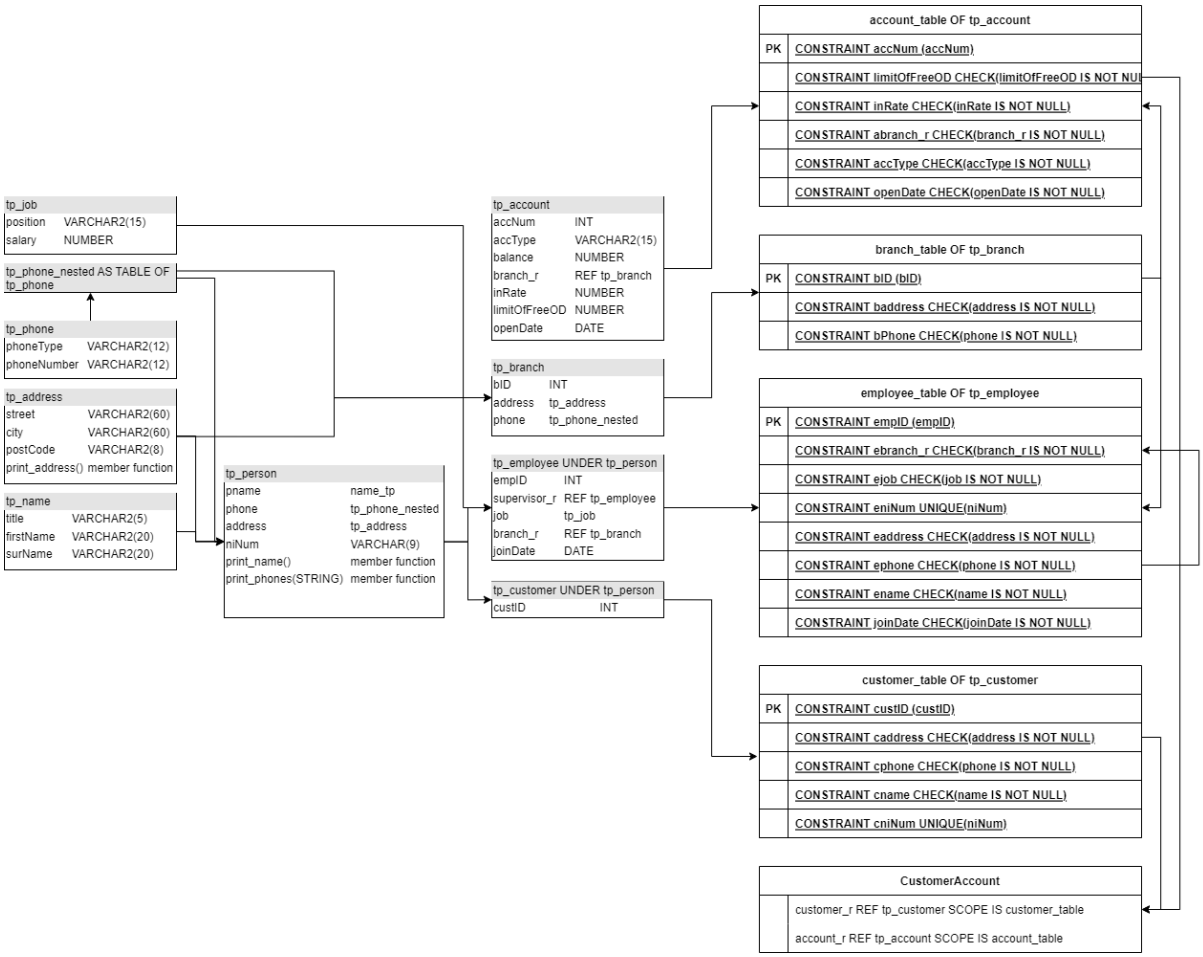
Task 1:



Task 2:

Object-oriented databases are a type of database that stores data in objects, like object-oriented programming languages. The object-relational approach to data modelling allows for some extra features that can help with modelling complex data and the relations between them. User defined structured types and inheritance of these types allow data to be modelled more naturally like it is structure in real life. Methods allow for functions to be defined within types which can help with operations that could e cumbersome to implement within an SQL query. References allow a row of another table to be “inserted” into a table, which allows the referenced data to be accessed from that object with the use of dot function. Constraints can still be added to tables like with an entity relational database allowing for control of what is inserted into the tables, defining the rules of the table. This can also be used on a reference to control what table the reference can be from. Collections allow for grouped data like lists to be added to a table which can enable an unknow number of entries to be inserted with the use of null or empty entries.

The database has been redesigned to take advantage of these object relational features while capturing the semantics of the original database design; an E-R diagram of the redesigned database is found below.



Structured types

I have used structured types to better organise the data from the original database.

tp_name: The data associated with a person's name has been separated into a name object containing the title, firstName and surName.

tp_address: The address data has been separated any address data into an address object containing the street, city, and postCode, this allows for the address object to be used for a person or for a branch as it is the same data used in both tables.

tp_job: The job has been encapsulated in a type containing the position and salary of an employee.

tp_phone: Each phone number has been changed to an object containing the phoneType and phoneNumber.

tp_phone_nested: Instead of having multiple attributes for each phone number in a table I have used a nested table of tp_phone.

tp_branch: the data from the branch_table has been encapsulated into a type to allow for referencing from other objects. It contains the bID, tp_address, and tp_phone_nested.

tp_person: The person type contains the common data from the customer an employee table containing the name, address, and phone types. It contains tp_name, tp_address, tp_nested_phone, and niNum.

tp_customer: The customer type inherits from the person type meaning all attributes contained in the person type is contained in the customer type. This type adds the custID which is the unique identifier for each customer.

tp_employee: This type also inherits from tp_person. This builds upon the person type by adding the empID, supervisor_r which is a reference to another employee, tp_job branch_r which is a referench to a tp_branch, and joinDate.

tp_account: Each entry from the account table will also be contained in an object containing the accNum, accType, balance, branch_r,inRate, limitOfFreeOD, and openDate.

This object-oriented approach encapsulates the data in a way that we do in real life, providing a more natural organisation of the database. I have also made each table be a table of types which will allow for references to be used if the database was extended in the future. This structuring of types promotes code reuse and allows for better maintainability of the database as if for example a change to how addresses are stored had to be made in the future the database would only have to be altered in one place instead of many.

Inheritance

Inheritance has been used to clean up the customer and employee tables. The data that is common between the two tables has been placed into the person object, which has been extended to customer and employee object which contain the rest of the data for their respective tables. This allows for both objects to share the code such as member functions contained in the person object, without having to implement them for both separate objects.

References

References has been used in the employee table and account table to reference the branch that the employee works at or the branch that the account is managed by. This allows for the details of the branch to be accessible from an entry in these tables without using joins with foreign keys. Another reference was used in the employee_table which references another employee that is their supervisor, this allows for the supervisor’s details to be accessible through the employee. References has also been used in the customerAccount table to reference a customer in the customer table and the account of the customer in the account table. This allows for all the data to do with customers and accounts to be accessible through one table, providing an entry point to query all data without using multiple joins. The use of references in the redesigned database allows for better accessibility of data without the need of using joins in the SQL query, making it easier to retrieve data.

Methods

I have added 4 member functions to the redesigned database to help with the queries in section 4.

The first member function is the print_address() function which is contained within the address type. This prints the address as a single string instead of having to specify each column in the query. Having it in the address type allows for is to be called from a branch or from a person as they both use the address withing their types.

The next function is the print_name() function, within the person type. This function prints the title. First name and surname of a person as a single string. It is available from either the employee type or the customer type as they both inherit from the person type. This function makes it easier to retrieve a person's name when querying the database.

The 3rd member function is the print_phones(type) function contained within the person type. This function takes a string as a parameter which represents the type of phone, they want to retrieve the numbers of. It then returns all the numbers that the corresponding type matches the string passed to it as a single string. This makes it easier to retrieve all the phone numbers of a person from a single SQL query.

The last member function included in the redesigned database is the award_medal() function. This function finds out how many employees a person supervises and how long they have worked at the bank. It then uses this data to determine what star they will receive based on the parameters of Task 5 H. The function then returns a string representing what star the employee will receive either gold, silver , or bronze.

These member functions make it easier to retrieve data that is commonly queried together, making it simpler to construct a query. The award_medal() provides practical functionality for the bank which evaluates their employees based on the factor they have determined.

Constraints

Constraints have been added to each table that preserves the primary keys from the original database. This makes sure that they are both unique and required to be included in each entry to the table. Many columns of each table also have a constraint to check they are not NULL as since this is for a bank much of the data will be essential to the operations of the bank. The national insurance numbers of every person have a constraint which specifies that it must be unique, mirroring the functionality of NI numbers in real life. There are also constraints added to the customerAccount table which specifies what table the references can be from. This makes sure that a reference to a customer, points to an entry in the customer_table and a reference to an account, points to an entry in the account_table. Further constraints could have been added which restricts the data that is entered into the table, for example for the accType I could have specified only "Savings" or "Standard" could be added to the table. I decided against this as I wasn't sure it provided any benefit in this application since this may not be needed in a real-life scenario, where data will most likely be entered into the database using a GUI which would achieve the same thing as the constraint with a drop-down menu which would be easier to maintain. Constraints allow for

the data entered into the database to be controlled further than just the types defined in each attribute and can provide extra security for a database.

Collections

Collections has been used for storing the phone numbers of contained in the person type and the branch type. I have used nested tables to store this data as we can not be certain how many phone numbers an entry will have, some people may have more than one mobile number or not have a house phone. A nested table is an unordered list that is inserted as an attribute in a table. Each entry into the nested table contains two VARCHAR2 one for the type of phone and the other for the phone number. I could have had a type that contained the house, mobile1, and mobile2 which was specified in the original schema, but this would contain NULL entries for people that only had 1 mobile or no house phone. Another approach I could have used is Varrays to store the phones, however the maximum size must specified which could limit functionality if extra phones are unable to be added. For example, a work mobile or internal phone number may be needed to be entered to the database for employees but can't since the Varray was too small.

Task 3:

Database implementation found in DBCreating.sql.

Test data found in DBPopulating.sql.

Task 4:

Complete script with queries found in AnswersToTask4.sql

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

```
---select the names and addresses
SELECT e.print_name() AS name, e.address.print_address() AS address
FROM employee_table e
---where first name contains 'st' and lives in edinburgh
WHERE UPPER(e.name.firstName) LIKE '%ST%'
AND e.address.city = 'Edinburgh';
```

Output:

	NAME	ADDRESS
1	Mr. Christopher Brown	789 Elm Street, Edinburgh, EH6 5CD
2	Mr. Steven Stewart	1212 Elm Street, Edinburgh, EH5 6YZ

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

```
---select address of branch and number of accounts
SELECT a.branch_r.address.print_address() AS address,
    count(a.accType) AS "number of savings accounts"
FROM account_table a
---where accType is savings
WHERE UPPER(accType) = 'SAVINGS'
GROUP BY a.accType, a.branch_r.address.print_address();
```

Output:

	ADDRESS	number of savings accounts
1	30 George Street, Glasgow, G1 2PA	3
2	456 Sauchiehall Street, Glasgow, G2 3JD	2
3	142 Lothian Road, Aberdeen, AB11 5BA	3
4	49 Hanover Street, Dundee, DD1 3DQ	2
5	30 St Andrew Square, Inverness, IV1 1EX	3
6	100 George Street, Prestonpans, EH32 9BG	3
7	80 George Street, Haddington, EH41 3QS	5
8	68-72 Rose Street, Haddington, EH41 3QS	3
9	15-17 South St Andrew Street, Edinburgh, EH12 3AB	3
10	2 Rutland Square, Edinburgh, EH1 2AS	3
11	30-34 North Bridge, Glasgow, G1 1QN	3
12	12-14 Frederick Street, Glasgow, G2 2HB	3
13	35-37 Leith Street, Glasgow, G3 3AT	3
14	36 Nicolson Street, Edinburgh, EH8 9DT	3
15	6-7 South St Andrew Street, Edinburgh, EH2 2AZ	1

c.) At each branch, find customers who have the lowest balance in their savings account, displaying the branch ID, the customer's full names, and the balance.

```
---select, bid, full name, and balance
SELECT c.account_r.branch_r.bID AS bID,
    c.customer_r.print_name() AS name,
    c.account_r.balance AS balance
FROM (
    ---select the minimum balance from each branch
    SELECT c.account_r.branch_r.bID AS bID,
        MIN(c.account_r.balance) AS balance
    FROM CustomerAccount c
    WHERE c. account_r.accType = 'Savings'
    GROUP BY c.account_r.branch_r.bID, c.account_r.accType
    ) balance, customerAccount c
---where the minimum balance matches the minimum balance of the branch
WHERE c.account_r.branch_r.bID = balance.bID
AND c.account_r.balance = balance.balance;
```

Output:

	BID	NAME	BALANCE
1	1	Ms. Sarah Morrison	2200
2	2	Mr. David Clark	4000
3	3	Mr. Jack Grant	3800
4	4	Mrs. Connie Grant	4100
5	5	Mrs. Victoria Walker	4300
6	9	Mr. John Stevenson	5100
7	14	Mr. Daniel Stevenson	6100
8	7	Mr. Michael Mackenzie	4700
9	6	Mr. Darren Smith	4500
10	6	Mrs. Martha Smith	4500

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 name, banks they work at, and bank they are a customer at
SELECT e.print_name() AS name,
    e.branch_r.address.print_address() AS "Employed at",
    c.account_r.branch_r.address.print_address() AS "Banks with"
FROM employee_table e
---inner join customer_table and account_table on niNum
---which returns the people that works at the banks and have an account in the
bank
INNER JOIN customerAccount c
ON e.niNum = c.customer_r.niNum
---where the employees' supervisor is a manager
WHERE e.supervisor_r.job.position LIKE '%Manager';
```

Output:

	NAME	Employed at	Banks with
1	Ms. Sarah Morrison	30 George Street, Glasgow, G1 2PA	30 George Street, Glasgow, G1 2PA
2	Mr. David Clark	30 George Street, Glasgow, G1 2PA	456 Sauchiehall Street, Glasgow, G2 3JD
3	Mr. Jack Grant	142 Lothian Road, Aberdeen, AB11 5BA	142 Lothian Road, Aberdeen, AB11 5BA
4	Ms. Emma Murray	142 Lothian Road, Aberdeen, AB11 5BA	142 Lothian Road, Aberdeen, AB11 5BA
5	Mrs. Emily Smith	30 George Street, Glasgow, G1 2PA	15-17 South St Andrew Street, Edinburgh, EH12 3AB
6	Mr. Christopher Brown	15-17 South St Andrew Street, Edinburgh, EH12 3AB	15-17 South St Andrew Street, Edinburgh, EH12 3AB
7	Mr. Matthew Street	15-17 South St Andrew Street, Edinburgh, EH12 3AB	15-17 South St Andrew Street, Edinburgh, EH12 3AB
8	Mr. Steven Stewart	15-17 South St Andrew Street, Edinburgh, EH12 3AB	36 Nicolson Street, Edinburgh, EH8 9DT

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 the joint current account.

```
---select the bid, account holder names, and their overdraft
SELECT maxJoint.bID as bID,
    MIN(c.customer_r.print_name()) || ', ' || MAX(c.customer_r.print_name())
AS "Account Holders",
    maxJoint.OD AS OD
FROM (
    ---select each entry that the OD matches the maxOD from subquery
    SELECT bID,
        accNum,
        OD
    FROM (
        ---add row that contains the highest OD jointAccount from the branch
        SELECT jointAccounts.bID AS bID,
            jointAccounts.accNum AS accNum,
            jointAccounts.OD AS OD,
            MAX(jointAccounts.OD) OVER (PARTITION BY jointAccounts.bID) AS
maxOD
        FROM (
            ---select all accounts with more than 1 holder
            SELECT c.account_r.accNum AS accNum,
                c.account_r.branch_r.bID AS bID,
                MAX(c.account_r.limitOfFreeOD) AS OD,
                COUNT(c.account_r.accNum) AS accountHolders
            FROM customerAccount c
            GROUP BY c.account_r.accNum, c.account_r.branch_r.bID
            HAVING COUNT(c.account_r.accNum) > 1) jointAccounts
        )
    WHERE OD = maxOD) maxJoint, customerAccount c
WHERE c.account_r.accNum = maxJoint.accNum
GROUP BY maxJoint.bID, maxJoint.OD;
```

Output:

	BID	Account Holders	OD
1	15	Mr. George Gray, Mrs. Shaunna Gray	1900
2	6	Mr. Darren Smith, Mrs. Martha Smith	950

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

```
--select name, ID, and all mobile numbers
SELECT c.custID AS custID,
    c.print_name() AS name,
    c.print_phones('mobile') AS "mobile numbers"
FROM (
    ---select all customers that have more than 1 mobile
    SELECT c.custID AS custID,
    FROM customer_table c, table(c.phone) p
    WHERE p.phoneType = 'mobile'
    GROUP BY c.custID, p.phoneType
    HAVING COUNT(p.phonetype) > 1) multipleMobiles,
    customer_table c, table(c.phone) p
---where customer has multiple mobiles and at least one begins with '0750'
WHERE c.custID = multipleMobiles.custID
AND p.phoneNumber LIKE '0750%'
---group together in case a customer has more than one mobile beginning with
'0750'
GROUP BY c.custID, c.print_name(), c.print_phones('mobile');
```

Output:

	CUSTID	NAME	mobile numbers
1	1010	Mr. Cameron Jones	0779348899, 0750939348
2	1012	Mr. Connor Martin	07505432109, 07551423419
3	1013	Mrs. Evie Martin	07506662109, 07508938432
4	1034	Mrs. Annie Smith	07766848398, 07507897543

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

```
---select count of employees
SELECT COUNT(e.empID) AS "Employees supervised by Mrs Smith, who is supervised
by Mr. Barclay"
FROM employee_table e
---where supervisor is 'mrs smith'
WHERE e.supervisor_r.name.title = 'Mrs'
AND e.supervisor_r.name.surName = 'Smith'
---and supervisor's supervisor is 'mr barclay'
AND e.supervisor_r.supervisor_r.name.title = 'Mr'
AND e.supervisor_r.supervisor_r.name.surName = 'Barclay';
```

Output:

	Employees supervised by Mrs Smith, who is supervised by Mr. Barclay
1	11

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 10 staff; silver medals for employees who have been working at the bank for more than 8 years and supervised more than 6 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 employees name and award they will be given
SELECT e.print_name() AS name,
    e.award_medal() AS medal
FROM employee_table e
WHERE e.award_medal() IS NOT NULL
---order medals given by gold -> silver -> bronze
ORDER BY CASE WHEN e.award_medal() = 'Gold' THEN 1
              WHEN e.award_medal() = 'Silver' THEN 2
              ELSE 3 END;
```

Output:

	NAME	MEDAL
1	Mrs. Emily Smith	Gold
2	Mr. David Miller	Silver
3	Ms. Emily Stewart	Bronze
4	Ms. Linda Wilson	Bronze
5	Mr. David Johnson	Bronze
6	Ms. Sophia Miller	Bronze
7	Mr. Michael Brown	Bronze
8	Ms. Olivia Davis	Bronze
9	Mr. Matthew Street	Bronze
10	Ms. Rachel Wilson	Bronze
11	Mr. Ryan Campbell	Bronze
12	Mr. Connor Young	Bronze
13	Mr. Liam Campbell	Bronze
14	Mr. Steven Stewart	Bronze

Task 5:

The object-relational database serves the same purpose of storing data as the entity relational database, however there is advantages and disadvantages to this approach to data modelling.

Advantages

One of the main advantages of the object-relational approach is the ability to model data in a more natural way that resemble real-life situations, the use of structured types and inheritance can make a complex schema easier to understand on a human level. This modular approach also makes it easier to maintain and make changes to the database as the attributes are encapsulated within object instead of being entered into a table. Inheritance promotes polymorphism and code-reuse. These features of the object-relational approach have allowed the redesigned database to have an easier to understand structure with types that a human will be able to understand easily. Inheritance allows for tables that share common attributes to be encapsulated in an object which can then be used in a table, This can be seen in the redesigned database with the tp_employee and tp_customer types inheriting from the tp_person type. The relational database is not modular and cannot take advantage of inheritance which may mean that changes to multiple parts of the database will need to be made in the event of this happening.

Member functions provide a way to perform more complex operations on data within an object using the extended functionality provided by the PL/SQL programming language. This has been applied in the redesigned database , the print_name() and print_address() function allows for data that will commonly be retrieved together to be retrieved with a single "select" clause. This can make it easier to construct queries for the database. The print_phones(type) function utilises the looping function of PL/SQL to retrieve all the phone numbers of contained in the nested table where the type matches. The award() medal queries the database from within the member function, using that queried data to perform operations on the object it is called on. Member functions are aware of data types and objects defined in the data, adding more functionality to the function. Functions can be defined in an entity-relational database in the form of user defined functions however these are not associated with an object and cannot access attributes, they can only tasks on a dataset. User defined functions are more limiting and provide less functionality than the member functions in an object-relational database.

Another advantage is that the object-oriented approach can make it simpler to construct queries involving relations between entities. The use of references in tables allows for all data in an object to be retrievable without the use of a join, which can make retrieving data easier. This can be shown in the redesigned database with data from multiple tables being retrieved with a single "select" query, with the use of dot functions. The use of references replaces the primary key/foreign key relationship in the relational database which is easier to understand at first but can become more difficult as more complex relations are introduced, especially when querying the database.

Disadvantages

There are also some disadvantages to the object-oriented approach. One disadvantage is that object-relational databases can potentially have a larger storage size compared to entity-relational databases, due to storing additional information on types, relationships, and methods. Object-relational databases also have a lower adoption rate due to being more complex to construct. This leads to lesser resources and support being available to use with this approach. The object-relational approach may also impact performance with simple queries due to added complexity of the database, however there may be performance benefits with more complex queries involving relations between tables.

Conclusion

In conclusion I feel that the redesigned database is easier to understand as a human, the data is split up into logical chunks that are easier and more natural to comprehend. The added functionalities of the object-relational approach make it easier to maintain and simpler to extract data from the database. The use of references in particular makes it a much simpler task to construct queries for the database. I think the test data added is not sufficient to see any sort of difference in performance between the two approaches.

Task 6:

Script for dropping tables found in droppingTypesTables.sql