Database Concepts Assignment 2

Part A: Relational Database Design

Non-trivial FDs in `insurancePoliciy` relation,

```
InsurancePolicy (PolicyNumber, AgentNumber, PremiumAmount,
CoverageAmount)
```

1. **List all of the non-trivial FDs in the `InsurancePolicy` relation, including those implied by a key. Is the `InsurancePolicy` relation in BCNF? Explain briefly your answer. Note that it is possible for several, different insurance policies to have the same agent assigned (that is, to have the same agent number listed in the `AgentNumber` attribute). Does this kind of redundancy cause any update anomalies? Can we normalise this relation in order to remove this redundancy?**

`PolicyNumber` → `AgentNumber` ✓
`PolicyNumber` → `PremiumAmount` ✓
`PolicyNumber` → `CoverageAmount` ✓

The functional dependencies above are all in BCNF because the attributes on the left-hand side of the arrows is a key.

If you update the primary key in 'AgentNumber' in the Agent relation, then you have to update it multiple times in the InsurancePolicy relation where AgentNumber is being used as a foreign key, and this causes an update anomaly. Because several different InsurancePolicies can belong to one agent, this is a many-to-one relationship so we need another table sitting inbetween to normalize the data.

2. List all of the non-trivial FDs in the `Person` relation, including those implied by a key. Is the `Person` relation in BCNF? Explain your answer. If the `Person` relation is not in BCNF, decompose the `Person` relation into relations that are in BCNF

`Id` → `TFN` ✓
`Id` → `Name` ✓
`Id` → `Phone` ✓
`TFN` → `Id` ✓

The values on the left-hand side are keys so therefore all FDs are in BCNF.

3. List all of the non-trivial FDs in the PolicyOwner relation, including those implied by a key. Which normal form is this relation in? If it's not in BCNF, decompose the PolicyOwner relation into BCNF.

PolicyNumber, TFN → Start-Date ×
TFN → PersonID ×
PersonID → TFN ×

The above functional dependencies are all part of a key or are a candidate key but it is not the primary key. Therefore these all violate BCNF. To fix this I need to decompose into 3NF and to do this I first find the minimal basis.

PersonID, PolicyNumber → Start-Date

This gives me the new relation
PolicyOwner (PersonID*, PolocyNumber*, Start-Date)

Part B: SQL

1. **Give the bookdescID of books by authors whose surname starts with "S" or by publishers whose name starts with "S".**

```
SELECT book.bookdescID--, author.lastname,
publisher.publisherfullname
FROM book, written_by, author, publisher, published_by
-- join to author table
WHERE book.BOOKDESCID = written_by.BOOKDESCID
AND written_by.authorID = author.authorID
-- join to publisher table
AND book.BOOKDESCID = published_by.BOOKDESCID
AND published_by.publisherID = publisher.publisherID
-- find names that start with S
AND author.lastname like 'S%' OR publisher.publisherfullname like
'S%';
```

2. **With which publisher(s) the author Alfred Aho published his book(s)? Display publishers' full names. You must not use any subquery.**

```
SELECT publisher.publisherfullname --, author.FIRSTNAME,
author.LASTNAME
FROM publisher, published_by, written_by, author
WHERE publisher.PUBLISHERID = published_by.PUBLISHERID
AND published_by.BOOKDESCID = written_by.BOOKDESCID
AND written_by.AUTHORID = author.AUTHORID
AND author.FIRSTNAME  = 'ALFRED'
AND author.LASTNAME = 'AHO'
```

3. **Who are the authors that published books with the MC GRAWHILL publisher? Display the firstname and lastname of these authors.**

```
SELECT FIRSTNAME, MIDDLENAME, LASTNAME
FROM author, publisher, published_by, written_by
WHERE author.AUTHORID = written_by.AUTHORID
AND written_by.BOOKDESCID = published_by.BOOKDESCID
AND published_by.PUBLISHERID = publisher.PUBLISHERID
AND publisher.PUBLISHERFULLNAME = 'MC GRAW-HILL';
```

4. **Display the first name and lastname of authors who wrote more than 3 books. Along with each name, display the number of books as well.**

```
SELECT author.FIRSTNAME, author.LASTNAME, COUNT(book.BOOKDESCID)
AS totalBooks
FROM ((book
INNER JOIN written_by ON book.BOOKDESCID = written_by.BOOKDESCID)
INNER JOIN author ON author.AUTHORID = written_by.AUTHORID)
HAVING COUNT(book.BOOKDESCID) >= 3
GROUP BY author.FIRSTNAME, author.LASTNAME;
```

5. **Display the title of the book which has the largest number of physical copies. If there are more than one book with the largest number of copies, show them all. Your query should show the number of copies along with the title.**

```
-- join the book title to the subquery
SELECT book.title, COUNT(book_copy.BOOKDESCID)
FROM (book JOIN book_copy ON book.BOOKDESCID = book_copy.BOOKDESCID)
GROUP BY book.title
HAVING COUNT (book_copy.BOOKDESCID) =
-- subquery to get the maximim of the count
(SELECT  MAX(COUNT(book_copy.BOOKDESCID)) AS total_copies
FROM (book JOIN book_copy
ON book.bookdescID = book_copy.bookdescID)
GROUP BY book.title);
```

Part C: ER Model

Map the ER diagram below into a relational database schema. For each relation schema, underline the primary key and denote any foreign key with an asterisk (*).

1. **Entites**
   Account (AcctNo, balance)
   Customer(custID, c-fname, c-lname)
   Product(pro-code, name)
   Employee(empID, e-fname, e-lname)
   Branch(branchCode, address)

2. **Weak Entities**
   Dependent(segNo, AcctNo*, custID*, d-fname, d-lname, shcool)

3. **Relationships**
   Own (Acctno*, custID*)
   Work (empID*, branchCode*)

4. **Ternary Relationships**
   Promotion(custID*, pro-code*, empID*, outcome)

Part D. Research Questions

1. The Library database schema in Part B assumes that all books borrowed in one transaction are to be returned together. However, this is an unreasonable assumption and quite contrary to the common practice across all forms of libraries. You can return books (borrowed together) separately. To accommodate this requirement, discuss what relation schemas need to be changed and give the new relation schemas.

The 'due date' and 'return date' of each individual book can be move from the borrow table to the borrow_copy table. This way you can determine when each individual book in a borrow transaction is due back and actually returned. The new schema will look as following.

```
borrow(transactionID, personID*, borrowdate)
borrow_copy(transactionID*, bookID*, duedate, returndate)
```

None of the attributes that have been moved are keys and the transaction ID key is consistent between the two relations so this modification should not do much damage to the design of the database.

Depending on the borrowing rules of the library however, the time length of each borrow might not vary between books. If this is the case then `duedate` can stay in the borrow relation to prevent redundancy.

2. Discuss if the Library database schema in Part B allows customers to extend their loans before the due date. If so, explain how loan extensions are likely handled and any limitations with the approach.

To extend the loan of a book, it would require another transaction in the borrow relation. This means that when a book is returned, it would have to update the `duedate` for multiple transactions causing an update anomaly.

What you could do is set the return date to the date the loan is extended. Again, if they are extending the loan for only one book then `returndate` will need to be moved to the `book_copy` relation.

The limitations of this are that only one type of transaction that can be made. If you need to differentiate between the two then you could create another relation for re-borrow that includes the date the books were re-borrowed and the new `duedate`. This of course would mean that the `duedate` also has to be updated in the borrow relation.