Database Assignment 2

Part A:

1) Functional Dependencies:

Department:

deptID : deptName, manager, empID

Employee:

empID -> empName, deptID, email

email : empID, empName, deptID

Project:

projID : startYear, deptID

EmpProj :

projID, role : empID

empID, projID: role

Evaluation:

projID, evalDate : rating

projID : manager

2.

Schema :- Department(deptID, deptName, manager*, empID*)

Candidate keys: NONE

Primary key: deptID

Foreign key: manager(Employee.empID), empID(Employee.empID)

- It is not in first normal form. One department may have more than one employee, according to the business rule. Then, instead of potentially keeping numerous different employee IDs, just one employee ID.The department table's personnel IDs defy the first conventional form.

Schema :- Employee(empID, empName, deptID*, email)

Primary key : empID

Candidate key : email

Foreign key : deptID(Department.deptID)

- It is in the Third normal form as all values are atomic and the non-key attributes depend on the whole key and there is no transitive dependency.

Schema :- Project(projID, startYear, deptID*)

Primary key : projID

Candidate key : None

Foreign key : deptID(Department.deptID)

- It is in the Third normal form as all values are atomic, the non-key attributes depend on the whole key, and there is no transitive dependency.

Schema :- EmpProj(empID*, projID*, role)

Primary key : (projID, role)
Candidate key : (empID, projID)
Foreign keys : empID(Employee.empID), projID(Project.projID)

- Due to the atomic nature of all values, the non-key attributes' dependence on the entire key, and the absence of transitive dependencies, it is in the third normal form.

Schema :- Evaluation(projID*, manager*, evalDate, rating)

Primary key : projID,evalDate

Candidate key : None

Foreign keys : projID(Project.projID), manager(Department.manager)

- It is in the First normal form as all values are atomic.
- Due to the 'manager' foreign key's independence from the 'evalDate' of the composite primary key, it is not in the second normal form. The projID is used to determine who is the "Manager."

Q.3. Ans

The relations not in third normal form

Department(deptID, deptName, manager*, empID*)
Evaluation(projID*, manager*, evalDate, rating)

A. Department(deptID, deptName, manager*, empID*)

The department table's empID foreign key is unnecessary. One department may have more than one employee, under the business rule. Thus, keeping a single employee ID in the department table when there may be numerous different employee IDs violates the first normal form. This foreign key attribute is deleted. Since the deptID is the foreign key in the employee table, there is no data loss. The third normal form's department schema will be:

- Department(deptID, deptName, manager*)
  DepartmentEmployee(deptID*, empID*)

B. Evaluation(projID*, manager*, evalDate, rating)

manager* is an unnecessary foreign key in the Evaluation table and it doesn't depend on the whole primary key. We can remove the manager attribute from this table to get it into the third normal form. There is no need to create a separate table to associate managers with the project they reviewed as each project belong to a singular department and each department has a singular manager. The evaluation schema in the third normal form will be

Evaluation(projID*, evalDate, rating)

ProjectEvaluator(projID*, manager*)

4. The current schema is as follows.

Department(deptID, deptName, manager*)
DepartmentEmployee(deptID*, empID*)
Employee(empID, empName, deptID*, email)

Project(projID, startYear, deptID*)
EmpProj(empID*, projID*, role)
Evaluation(projID*, evalDate, rating)
ProjectEvaluator(projID*, manager*)

The relation given by the schema, Employee(empID, empName, deptID*, email) and DepartmentEmployee(deptID, empID*) can be combined.ProjectEvaluator(projID*, manager*) is the same. Additionally, the transitive functional dependency provided by the projID and manager provides access to the relationship between the two.projID -> deptID -> manager

Therefore we can delete the ProjectEvaluator(projID*, manager*) schema as it redundant data.

The new and updated schema will be as follows :

Department(deptID, deptName, manager*)
Employee(empID, empName, deptID*, email)
Project(projID, startYear, deptID*)
EmpProj(projID*, role, empID*)
Evaluation(projID*, evalDate, rating)

PART B:

Q1. Ans

```
SELECT FIRSTNAME || ' ' || LASTNAME AS Name, ADDRESS || ', ' || CITY AS ADDRESS

FROM person

WHERE CITY = 'Portland';
```

Q2 .Ans

```
SELECT subject.subjectid, COUNT(book.BOOKDESCID) AS 'Total no. of books'

FROM subject
```

```
LEFT JOIN book
   ON subject.SUBJECTID = book.SUBJECTID
GROUP BY subject.SUBJECTID
ORDER BY 'Total no. of books' DESC;
```

Q3.a.

```
SELECT DISTINCT person.FIRSTNAME, person.LASTNAME, person.CITY
FROM person, borrow
WHERE person.personid = borrow.personid;
```

Q3.b

```
SELECT DISTINCT person.FIRSTNAME,  person.LASTNAME, person.CITY
FROM person
JOIN borrow
   ON person.personid = borrow.personid;
```

Q3. c

```
SELECT DISTINCT FIRSTNAME, LASTNAME, CITY
FROM person
WHERE personid IN (
                SELECT personid FROM borrow
               );
```

Q4

```
SELECT DISTINCT bk.bookdescID AS 'Book Desc', bk.title AS 'Book Title'
FROM book bk JOIN book_copy bkc
            ON bk.bookdescID = bkc.bookdescID
      JOIN written_by w
         ON bk.bookdescID = w.bookdescID
```

```
    JOIN subject sub
      ON bk.subjectID = sub.subjectID
WHERE sub.subjecttype = 'Databases'
GROUP BY bk.bookdescID, bk.title
HAVING COUNT(DISTINCT w.authorID) > 2;
```

Q5

```
SELECT title AS "Book Title",
firstname || ' ' || lastname AS "Borrower Name",
date(br.returndate) AS "Date of Return",
date(br.duedate) AS "Due Date",
julianday(br.returndate) - julianday(br.duedate) AS "Days Delayed"
FROM book b
        JOIN book_copy bc
                ON b.bookdescid = bc.bookdescid
       JOIN borrow_copy br_c
              ON bc.bookdescid = br_c.bookid
        JOIN borrow br
                ON br_c.transactionid = br.transactionid
        JOIN person p
                ON br.personid = p.personid
WHERE br.returndate > br.duedate
ORDER BY b.title;
```

Q6

```
SELECT b.bookdescid AS 'Book description id', b.title AS title, b.year AS year
 FROM book AS b
 WHERE b.bookdescid NOT IN (
                    SELECT bc.bookdescid
                      FROM book_copy bc JOIN borrow_copy AS boc
                        ON bc.bookid = boc.bookid JOIN borrow AS bo
                          ON boc.transactionid = bo.transactionid
                   )
 ORDER BY title ASC, year DESC;
```

Q7

```
SELECT author.firstname, author.lastname, written_by.role, book.title
   FROM author JOIN written_by
     ON author.authorID = written_by.authorID
```

```
JOIN book
   ON written_by.bookdescID = book.bookdescID
WHERE written_by.role = 'Author'
AND author.authorID IN (
                    SELECT authorID
                    FROM written_by
                    WHERE role = 'Author'
                    GROUP BY authorID
                    HAVING COUNT(DISTINCT bookdescID) > 1
                  )
ORDER BY author.lastname, author.firstname;
```

Q8

```
SELECT b.TITLE
   FROM book b
   WHERE UPPER(b.TITLE) LIKE '%NETWORK%'
   AND NOT EXISTS (
               SELECT 1 FROM written_by w
               WHERE w.BOOKDESCID = b.BOOKDESCID
               AND w.AUTHORID NOT IN (
                                 SELECT a.AUTHORID
                                 FROM author a
                                 WHERE a.LASTNAME IN ('Miller', 'Noel')
                                 AND a.FIRSTNAME IN ('Tim', 'Jason')
                               )
             )
   GROUP BY b.TITLE;
```

Q9

```
SELECT DISTINCT b2.title, a.firstname || ' ' || a.lastname AS 'Author Name', b2.year
   FROM book b1 JOIN written_by wb1
      ON b1.bookdescid = wb1.bookdescid
   JOIN author a
      ON wb1.authorid = a.authorid
   JOIN written_by wb2
       ON a.authorid = wb2.authorid
   JOIN book b2
       ON wb2.bookdescid = b2.bookdescid
   WHERE b1.title = 'COMPUTER SCIENCE' AND b1.bookdescid <> b2.bookdescid
   ORDER BY a.lastname, a.firstname, b2.year DESC;
```

Q10

```
SELECT pr.firstname || ' ' || pr.lastname AS 'Borrower', bk.title AS 'Book Title', sub.subjecttype AS 'Book
Subject', strftime('%Y-%m-%d', br.borrowdate) AS 'Borrow Date', strftime('%Y-%m-%d', BR.returndate) AS
'Return Date'
  FROM borrow AS br JOIN borrow_copy AS bwcy
        ON br.transactionID = bwcy.transactionID
     JOIN book_copy AS bkcy
        ON bkcy.bookID = bwcy.bookID
      JOIN book AS bk
        ON bkcy.bookdescID = bk.bookdescID
      JOIN subject AS sub
        ON bk.subjectID = sub.subjectID
      JOIN person AS pr
        ON br.personID = pr.personID
  WHERE sub.subjecttype LIKE '%Image Processing%'
  ORDER BY br.borrowdate;
```

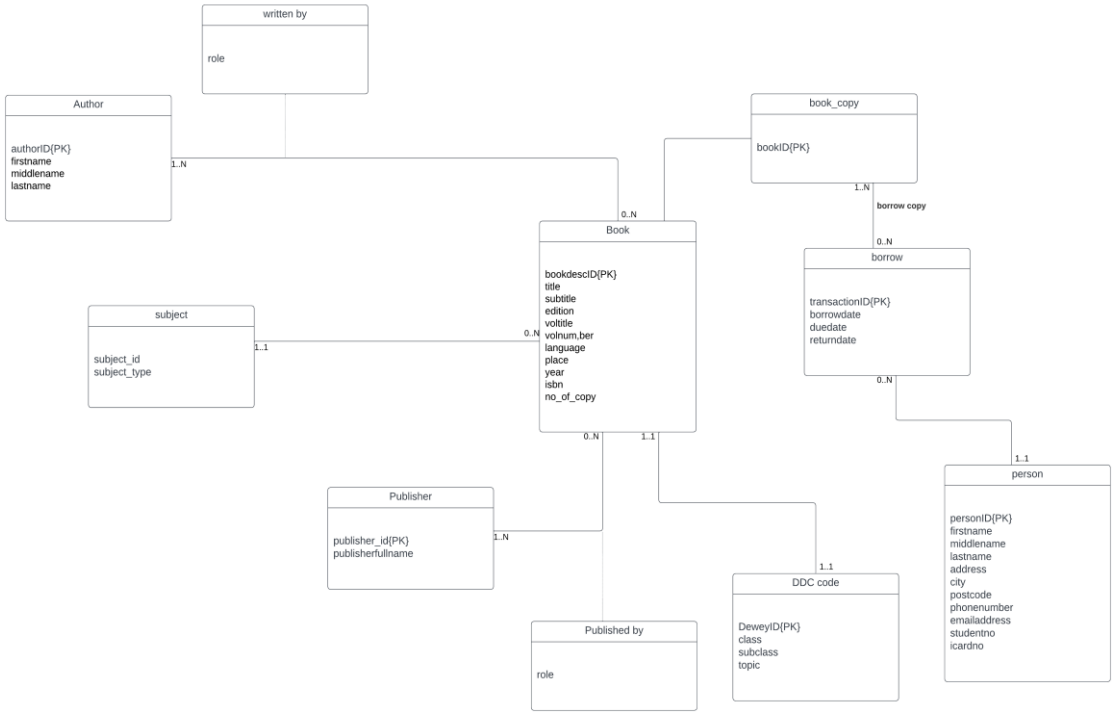
Part C: Research questions

1) Integrity constraints are guidelines that guarantee the authenticity, accuracy, and consistency of data in the database. Data cannot be applied to the Dewey Decimal Classification (DDC) on its own restrictions on the data's integrity. It offers a method for categorising the data into various groups and particular subgroups. Although the sequence in which the data is saved doesn't matter when using databases. The Dewey Call number cannot be used as the primary key in the book table since it is possible for two books in the library database to have the same Dewey Call number. Applying an integrity restriction by designating the topic id as the Dewey call number is one method. To link the book to the subject it pertains to and to preserve referential integrity, the Book table can use this Dewey call number as a foreign key. By doing this, the need for a separate column for the subject ID is removed, and every book receives a legitimate Dewey call number. However, this would also require grouping together and representing as a single subject a number of related subjects with the same Dewey call number. It is not something that the database can handle on its own to guarantee that the proper Dewey call number is given to each book. The Dewey Decimal Number would be the primary column in the modified subject table, which is as follows.

Dewey Decimal Number	Subject
005.265	Programming for specific computers
005.7	Data in computer systems

The book table will no longer have the subjectID column and the Dewey column will be a foreign key referencing the ‘Dewey Decimal Number’ primary key of the subject table.

Diagram:



2) Ans. As we can see, the person table has exactly the same information as the author table, with the exception that the person table's primary key is called "personId" rather than "authorId" in the author table. If we decided to keep the author's information in the Person table. We would avoid having to create a brand-new, distinct entity just for the Author. Additionally, because the person table already includes the extra fields, we will be able to record additional information on the author if necessary. According to the current author and person schema.
    - author(authorID, firstname, middlename, lastname)
    - person(personID, firstname, middlename, lastname, address, city, postcode, phonenumber,
    - emailaddress, studentno, idcardno)

    These two can be merged into a singular schema as the author is a subset of the person. As such

    the complete schema will be as follows

    borrow(transactionID, personID*, borrowdate, duedate, returndate)

    author(authorID, firstname, middlename, lastname)

    book_copy(bookID, bookdescID*)

    book(bookdescID, title, subtitle, edition, voltitle, volnumber, language, place, year, isbn, dewey,subjectID*)

    borrow_copy(transactionID*, bookID*)

    person(personID, firstname, middlename, lastname, address, city, postcode, phonenumber,

    emailaddress, studentno, idcardno)

publisher(publisherID, publisherfullname)

written_by(bookdescID*, authorID*, role)

published_by(bookdescID*, publisherID*, role)

subject(subjectID, subjecttype)

3) The following is how the scenario for a user borrowing books from the library is represented in the library database. Each book has two identifiers—one for the actual book and one for every copy of that same book. when taking out a book. The second ID is more important to the library since it allows it to monitor every copy of a specific book. A transactionID that is saved in the Borrow table along with the ID of the user who borrowed the book, the issue date, the due date, and the return date (if the book has been returned) is used to identify each transaction of a user borrowing a book. To track which books were borrowed in a specific transaction, the borrow_copy relation in the current library schema holds both the transactionID and the bookID. 'borrowDateTime' and 'PersonId' would have to be taken into account as the main key if the artificial primary key (transactionID) were absent, leading to a larger primary key. The borrow_copy table would then need to use this as well, resulting in the addition of three columns. This supports one of the drawbacks of utilising the Natural primary key, namely that it takes up more space. But if we were to do that, the updated schema for person, borrow and borrow_copy would be represented as follows

Initial Schema:

borrow(transactionID, personID*, borrowdate, duedate, returndate)
borrow_copy(transactionID*, bookID*) person(personID, firstname, middlename, lastname, address, city, postcode, phonenumber, emailaddress, studentno, idcardno)

Updated Schema : borrow(personID*, borrowdate, duedate, returndate)
borrow_copy(personID*, borrowdate*, bookID*) person(personID, firstname, middlename, lastname, address, city, postcode, phonenumber, emailaddress, studentno, idcardno)

References:

Natural vs Artificial Primary Keys. Available at: https://sqlstudies.com/2016/08/29/natural-vs-artificial-primary-keys