NAME : VIVEK AGGARWAL

STUDENT_ID : S4015465

DATABASE CONCEPTS

ASSIGNMENT 2

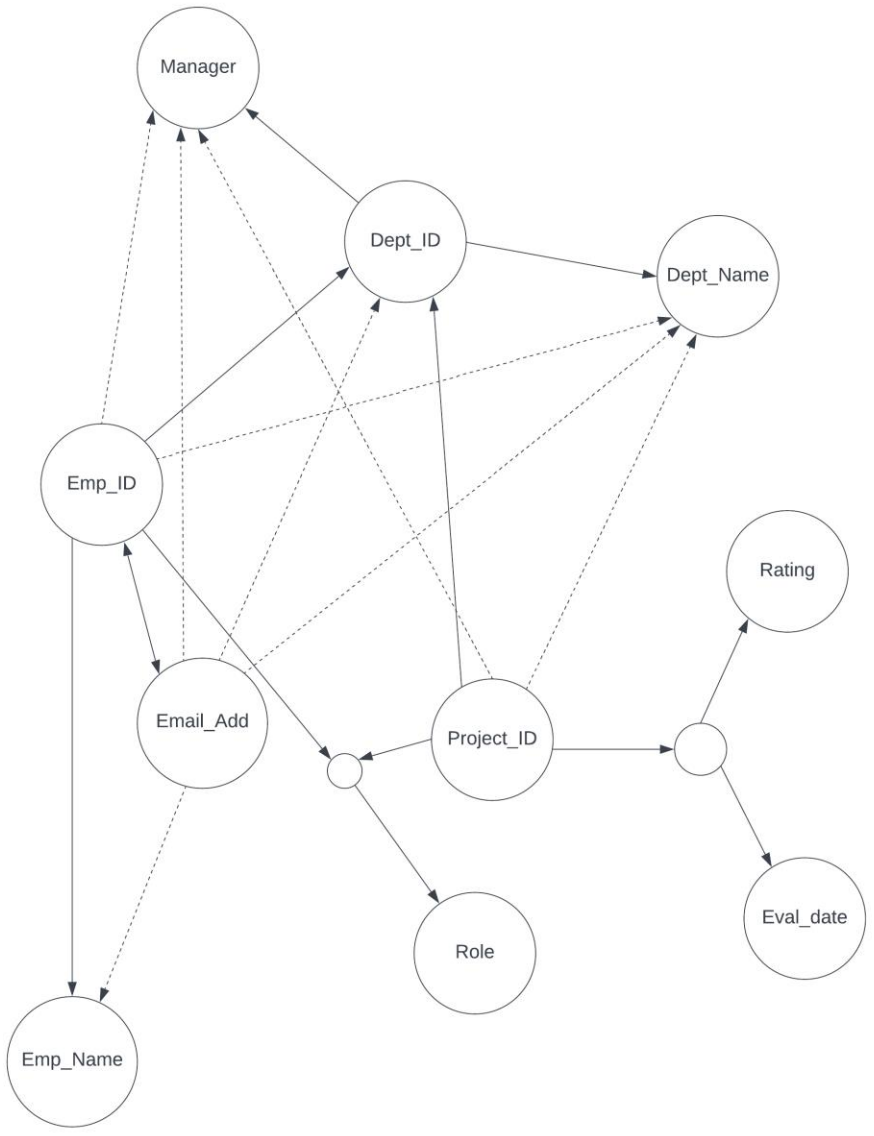
FINAL SUBMISSION

Task 1: Question-1

- Each employee belongs to only one department. Each employee has a unique email address.
- Each department has one manager and many employees.
- Each project is owned by one department.
- An employee can work on many projects with different roles, but has one role for each project.
- A project can have many employees and they can be from different departments.
- The progress of projects is evaluated once a week by department managers. The evaluation of a project on a particular date has a rating, which is an integer between 0 (very bad) to 5 (excellent).

ASSUMPTIONS:
-> Employee ID will also give us the employee name
-> Department ID will also give us the department name
-> Departments can exist that don't own any project
-> Manager gives us the manager of the department but its ID is also one of the employee ID

The following is the visual representation of the functional dependancies:



Functional Dependencies:

- Emp_ID <--> Email_add
- Emp_ID -> Dept_ID, Emp_name

- Dept_ID -> Manager, Dept_name
- Project_ID -> Dept_ID
- Emp_ID, Project_ID -> Role
- Emp_ID, Dept_ID -> Null
- Project_ID, Eval_date -> Rating

The transitive or redundant functional dependencies are as follows:

- Email_add -> Dept_ID
- Email_add -> Emp_name
- Emp_ID -> Manager
- Emp_ID -> Dept_name
- Email_add -> Manager
- Email_add -> Dept_name
- Project_ID -> Manager
- Project_ID -> Dept_name

Task 1: Question-2

1. Department(deptID, deptName, manager, <u>empID*</u>) 2NF

The emp_ID is the primary key in this relation as with emp_ID can give us dept_ID and with dept_ID we can get to dept_name and manager. Since we only need emp_ID to get to everything, it is the primary key of this relation.

Emp_ID -> Dept_ID

Dept_ID -> Manager, Dept_name

Emp_ID -> Manager, Dept_name

We can assume emp_ID is a foregn key in either this relation or in the relation below. But since Employees has a candidate key and is in 3NF, emp ID should be a foreign key in this relation.

There are no candidate keys in this relation as no other attribute can give us the emp_ID.

The given relation is in 2NF as everything depends on the whole key that is the primary key which is emp_ID. The given relation is not in 3NF as there are transitive dependencies present, like emp_ID -> dept_name, and emp_ID -> manager

2. Employee(<u>empID</u>, empName, deptID, email) 3NF

Assumption – 2 employees can have the same name, as there is no info given.

The emp_ID is the primary key in this relation as with emp_ID we can get to everything.

Emp_ID -> Dept_ID, Emp_name, email_add

Email is a candidate key in this relation, as it is unique and can also lead us to every other attribute in this given relation.

Email <--> emp_ID

The given relation is in 3NF as everything depends on just the primary key, and the whole primary key, and nothing else. Email is a candidate key, so we don't have to consider transitive dependencies for email. There are no other transitive dependencies present.

3. Project(<u>projID</u>, startYear, deptID)

Assumption: project_ID -> start_year

If we assume that project_ID gives us the start_year, then project_ID will be the primary key and the given relation will be in 3NF as everything would depend on the whole key and there wouldn't be any transitive dependencies among the given relation.

Project_ID -> Dept_ID

Then the relation would be Project(<u>projID</u>, startYear, deptID)

There are no candidate keys.

OR

The project_ID and the start_year are the primary keys in this relation. They together form a composite key. This is because we cannot get to start year by any way. The only functional dependency given is Project_ID -> Dept_ID. This means we cannot get start_year.

There are no candidate keys.

The relation would be Project(<u>projID</u>, <u>startYear</u>, deptID)

This relation is in 1NF as with the primary key, ie the composite key, can give us everything else. It is not in 2NF as everything does not depend on the full key. We can get to dept_ID from just project_ID and don't need start year for that.

4. EmpProj(<u>empID*</u>, <u>projID*</u>, role) 3NF

The primary keys for the given relation are emp_ID and project_ID. They together form the composite key for the given relation.

There are no candidate keys in this relation.

Emp_ID, Project_ID -> Role

The emp_ID is a foreign key as it is the primary key in one of the above relation Employee, which is in 3NF.

The proj_ID is also a foreign key as it is a primary key in the relation Project.

The given relation is in 3NF as everything depends on the whole key and there are no transitive dependencies in the given relation.

5. Evaluation(<u>projID*</u>, manager, <u>evalDate</u>, rating) 1NF

Proj_ID and the eval_date are the primary keys in the relation. If we have these 2, we can get to every other attribute present in the relation. The proj_ID gives us the manager and proj_ID combined with eval_date gives us the rating.

Project_ID -> Dept_ID

Dept_ID -> Manager

Project_ID -> Manager (transitive)

Project_ID, Eval_date -> Rating

Project_ID is a foreign key as it is a primary key in an above relation.

There are no candidate keys present in the relation.

The given relation is in 1NF as with the primary keys, we can get to every other attribute in this relation. It is not in 2NF as everything does not depend on the whole key. We can get to the manager with just the project_ID.

Task 1: Question-3

1. Department(deptID, deptName, manager, empID*) 2NF

Emp_ID -> Dept_ID

Dept_ID -> Manager, Dept_name

Emp_ID -> Manager, Dept_name

There is transitive dependency among the attributes, and manager, dept_name depend on dept_id that is not a key. It means emp_ID -> Manager, Dept_name is redundant. Therefore emp_id and manager and dept_name cannot be in a single table.
Hence, the new relations are:
Department 1.1(deptID*, empID*) 3NF
Department 1.2(deptID, deptName, manager) 3NF

Also emp_ID is a foreign key now as it is a primary key of Employee.

Dept_ID -> Manager, Dept_name

So Dept_ID is the primary key in department 1.2.

The new relations are now both in 3NF as everything depends on the full key and there are no transitive dependencies present.

2. Employee(empID, empName, deptID*, email) 3NF

Already in 3NF, but now dept_ID is a foreign key as it is a primary key in Department 1.2.

3. Project(projID, startYear, deptID*) 3NF

The dept_ID is a foreign key now as it is a primary key in Department 1.2.

Given relation already in 3NF

4. EmpProj(empID*, projID*, role) 3NF

The given relation is already in 3NF.

5. Evaluation(projID*, manager, evalDate, rating) 1NF

The given relation is in 1NF, as everything does not depend on the whole key.

Project_ID -> Dept_ID

Dept_ID -> Manager

Project_ID -> Manager (transitive)

Project_ID, Eval_date -> Rating

Manager has to go out of the relation as it does not depend on the whole key.

The new relations are:

Evaluation 1.1(projID*, evalDate, rating) 3NF

Evaluation 1.2(projID*, manager) 3NF

Task 1: Question-4

All the relations are:

Department 1.1(deptID*, empID*) 3NF
Department 1.2(deptID, deptName, manager) 3NF
Employee(empID, empName, deptID*, email) 3NF
Project(projID, startYear, deptID*) 3NF
EmpProj(empID*, projID*, role) 3NF
Evaluation 1.1(projID*, evalDate, rating) 3NF
Evaluation 1.2(projID*, manager) 3NF

The relation Department 1.1 and Employee will merge and form one relation as they have common primary key(emp_ID), we can can it Employees as no new attribute is being added into Employee. The new relation would be:

Employee(empID, empName, deptID*, email) 3NF

The relation Project and Evaluation 1.2 will merge as they have common primary key(proj_ID), and will form Project as follows :

Project(projID, startYear, deptID*, manager)

Now the given relation Project is no longer in 3NF as we have transitive dependency:

Project_ID -> Dept_ID

Dept_ID -> Manager

Project_ID -> Manager (transitive)

Hence project_ID and manager cannot be in the same table.

Therefore :

Project 1.1(projID, startYear, deptID*) 3NF
Project 1.2(deptID*, manager) 3NF

The relation Project 1.2 and Department 1.2 will merge as they have common primary key(dept_ID), and will form a new relation:

Department 1.2(deptID, deptName, manager) 3NF

The final relations are:

- Department(deptID, deptName, manager) 3NF
- Employee(empID, empName, deptID*, email) 3NF
- Project 1.1(projID, startYear, deptID*) 3NF
- EmpProj(empID*, projID*, role) 3NF
- Evaluation 1.1(projID*, evalDate, rating) 3NF

Renaming :

- Department (deptID, deptName, manager) 3NF
- Employee (empID, empName, deptID*, email) 3NF
- Project (projID, startYear, deptID*) 3NF
- Emp_role (empID*, projID*, role) 3NF
- Evaluation (projID*, evalDate, rating) 3NF

Task 2: SQL QUERIES

1. Display the name, and address of persons who live in the city of Portland. The output should have column headings "Name" and "Address", and respectively display first name and last name separated by a space, and address and city separated by comma and space. (104 rows)

```
SELECT firstname  || ' ' || lastname   AS Name,
        address   || ', ' || city AS Address
FROM person
WHERE LOWER(city) LIKE '%portland%';
```

2. Compute the total number of books for each subject. Output the subjectid together with its total number of books, in decreasing order of total number of books. (16 rows)

```
SELECT subject.subjectid AS SubjectID,
    COUNT (book.bookdescid) AS Total_books
FROM subject
LEFT JOIN book
    ON subject.subjectid = book.subjectid
GROUP BY subject.subjectid
ORDER BY Total_books DESC;
```

3. Display the first name, last name and city of persons who borrowed any book. There should not be any duplicate records in your result. (128 rows)
a. Write your query without using the JOIN operator keyword or any sub-query.

```
SELECT DISTINCT p.firstname, p.lastname, p.city
  FROM person AS p,
       borrow AS b
 WHERE p.personid = b.personid;
```

b. Write your query using the JOIN operator.

```
SELECT DISTINCT p.firstname, p.lastname, p.city
FROM borrow b
JOIN borrow_copy bc
    ON b.transactionid = bc.transactionid
JOIN person p
    ON b.personid = p.personid
```

c. Write your query using an IN sub query.

```
SELECT firstname, lastname, city
FROM person
WHERE personid IN (SELECT personid
                    FROM borrow)
```

4. Are there books that belong to on the subject 'Databases' that are written by more than 2 authors (in whatever role)? Display the bookdescid and title of these books. (0 rows)

```
SELECT DISTINCT b.bookdescid, b.title
FROM book b
INNER JOIN subject s
            ON b.subjectid = s.subjectid
INNER JOIN written_by w
            ON b.bookdescid = w.bookdescid
WHERE LOWER(s.subjecttype) LIKE 'databases'
GROUP BY b.bookdescid, b.title
HAVING COUNT(DISTINCT w.authorid) > 2;
```

5. The dates in relation "borrow" are stored as REAL type data in Julian Days. You can use the built-in date() function to convert the real value into date format. For example, to find out

a duedate as YYYYMM-DD format you can use date(duedate). Find out the borrowers who returned the books after the due dates. Display book title, “borrower name” (concatenated first name and last name), date of return, due date, and how many days were delayed from the due date. Display the dates in YYYY-MM-DD format. (33 rows)

```
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;
```

6. Find out the books that have never been borrowed. Display the book bookdescid, titles along with the year of publication, ordered alphabetically by title and then from newest to oldest. (295 rows)

```
SELECT b.bookdescid AS book_description_id,

       b.title AS Title,
       b.year AS "Year of Publication"
FROM book AS b
WHERE b.bookdescid NOT IN (SELECT bc.bookdescid
                             FROM book_copy bc
                             JOIN
                             borrow_copy AS c
                                 ON bc.bookid = c.bookid
                             JOIN
                             borrow AS b
                                ON c.transactionid =b.transactionid)
ORDER BY title ASC,
         year DESC;
```

7. Find out the authors (including co-authors) who have written (in the “Author” role) more than one book. Display the first name, last name of authors and their role along with the book title, ordered alphabetically by the last name and then first name of authors. (151 rows)

```
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 LOWER(written_by.role) LIKE '%author%'
AND author.authorid IN (SELECT authorid
                        FROM written_by
                        WHERE LOWER(role) LIKE '%author%'
                        GROUP BY authorid
                        HAVING COUNT(DISTINCT bookdescid) > 1)
ORDER BY author.lastname, author.firstname;
```

8. A person is reading books about Networks (title containing the string 'network' in any case). They want to find books on the topic co-written (in whatever role) by ONLY 'Tim Miller' and 'Jason Noel' (no other authors). Display the titles of these books. (0 rows)

```
SELECT b.title
FROM book AS b
       JOIN written_by AS wb
           ON b.bookdescid = wb.bookdescid
       JOIN author AS a
           ON wb.authorid = a.authorid
 WHERE b.title LIKE '%network%' AND
       a.firstname || a.lastname IN ('TimMiller', 'JasonNoel');
```

9. Find all OTHER books written by the author of the book with title ‘COMPUTER SCIENCE’. Display the title, “Author Name” (first name and last name), and year of publication of OTHER books by the same author(s). (1 row)

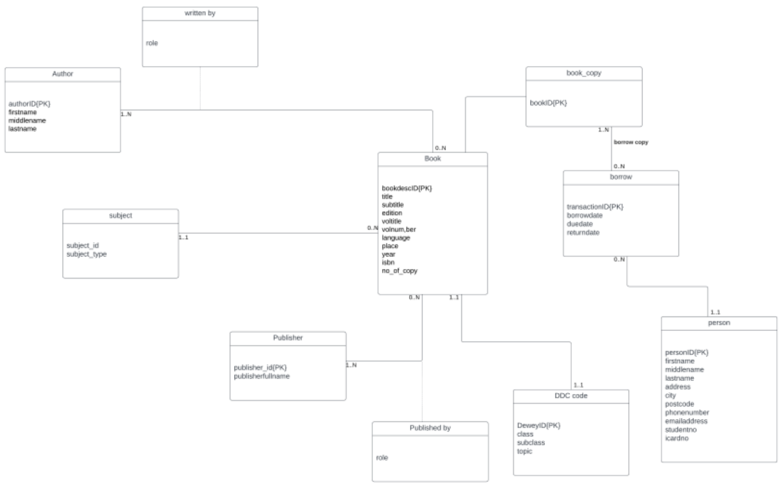
```
SELECT b.title,a.firstname || ' ' || a.lastname AS "Author Name",
b.year
FROM book b
JOIN written_by wb
     ON b.bookdescid = wb.bookdescid
JOIN author a
     ON wb.authorid = a.authorid
WHERE a.authorid IN ( SELECT authorid
                      FROM written_by wb
                      JOIN book b
                           ON wb.bookdescid = b.bookdescid
                      WHERE UPPER(b.title) LIKE 'COMPUTER SCIENCE')
AND UPPER(b.title) != 'COMPUTER SCIENCE';
```

10. Find out the persons who borrowed books on the subject of “Image Processing”. Display borrower name (first and last names, separated by a space) under the column heading of "Borrower", book title, book subject, borrow date, and return date. Borrow and return dates should be in the YYYY-MM-DD format. (5 rows)

```
SELECT p.firstname || ' ' || p.lastname AS "Borrower", b.title AS
"Book Title", s.subjecttype AS "Book Subject",
    strftime('%Y-%m-%d', br.borrowdate) AS "Borrow Date", strftime
('%Y-%m-%d', br.returndate) AS "Return Date"
FROM borrow br
JOIN borrow_copy bc
            ON br.transactionid = bc.transactionid
JOIN book b
            ON bc.bookid = b.bookdescid
JOIN subject s
            ON b.subjectid = s.subjectid
JOIN person p
            ON br.personid = p.personid
WHERE UPPER(s.subjecttype) LIKE '%IMAGE PROCESSING%'
ORDER BY p.lastname, p.firstname;
```

Task 3: REPORT

1. Data integrity restrictions are placed on the Dewey call number and the subject class of books by the Dewey Decimal Classification system. Each Dewey call number must follow the DDC structure, which consists of three-digit primary classes, two-digit hierarchical divisions, and one-digit particular subclasses. A book's major class and hierarchical division of its Dewey call number must also match the topic class of the book. The Dewey call number and its matching topic class are separated fields in the Book table in the provided Library database schema, which enforces this constraint. However, if the input is not vetted or confirmed, it is still possible for inaccurate data to be entered into the database. To verify that the data inserted into the database adheres to the DDC structure, the relational data model enables the application of data integrity measures like check constraints.



Consider a book with the following details:

- Title: "Introduction to Astrophysics"

- Author: John Smith
- Dewey call number: 520.3

The "DeweyClass" table would have the following entry:

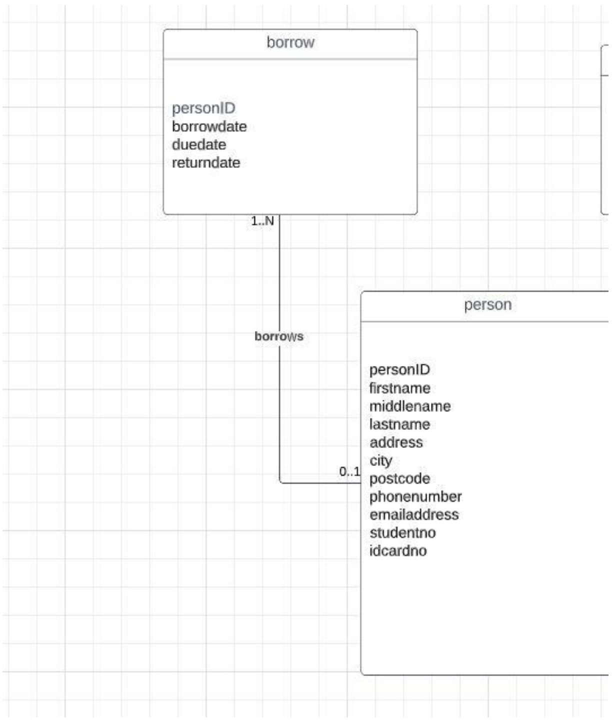
DeweyID	Class Name	Subclass
520	Astronomy and allied sciences	
520.3	Astrophysics	

BookID	Title	Author	DeweyID
1.	Introduction to Astrophysics	John Smith	520.3

The Dewey call number "520.3" in this instance relates to the subject class "Astronomy and allied sciences" (520) and the particular subclass "Astrophysics" (520.3). The Dewey call number is verified as valid and present in the "DeweyClass" database by the "DeweyID" foreign key in the "Book" table, which refers to the "DeweyID" primary key in the "DeweyClass" record.

2. Since a person can be both a library user and an author, the present design decision to separate Person and Author entity types without any relationship between them introduces a flaw in the ER model. By establishing a relationship between the Person and Author entity types, it is possible to enhance the ER model and relational database architecture by enabling people to be linked to the books they have written. Between the Person and Book entity types, this can be done as a many-to-many relationship with Author acting as an attribute.

3.



A transaction between a library user and a book is recorded in the Borrow table with the fictitious primary key "transactionID" in order to simulate the real-world scenario of "users borrow books" in the current Library database. Without the artificial primary key, the real-world situation would still entail a transaction between a book and a library user, but in order to uniquely identify each transaction, the primary key of the Borrow table would need to be made up of numerous properties. It would be necessary for the Borrow table to have a composite primary key made up of the User ID, Book ID, and Borrow Date entries. This could result in data fragmentation problems and difficult join queries when working with a lot of data. As a result, the usage of a fake primary key in this case facilitates the database design's simplification and enhances query performance.

REFERENCES:

- Natural vs Artificial Primary Keys | SQL Studies
- List of Dewey Decimal classes - Wikipedia