Assignment Cover Page

Course Name	ISYS – Database Concept
Student Name	Duy Phong Thach
Student Number	S3821853
Lecturer	Dr Halil Ali
Class number	1

PART A:

Question 1:
Functional dependencies for each relation:
a) Department:
deptID → deptName, manager
empID → deptID
empID → deptName, manager (transitive – redundant)

b) Employee:
empID → empName, deptID, email
email → empID, empName, deptID

c) Project:
projID → startYear, deptID

d) EmpProj:
empID, projID → role

e) Evaluation:
projID, evalDate → rating
manager, evalDate → rating
projID → manager

Question 2:

a) Department(deptID, deptName, manager*, empID*)

Primary key: deptID
Candidate key: deptID, empID
Highest normal form: It is not in normal form.
Reason: Single value is not in its own cells, because with each deptID there are many empID.

b) Employee(empID, empName, deptID*, email)
Primary key: empID
Candidate key: empID, email
Highest normal form: 2NF
Reason: Because the attributes also depend on non-primary key

c) Project(projID, startYear, deptID*)
Primary key: projID
Candidate key: projID
Highest normal form: 3NF

d) EmpProj(empID*, projID*, role)
Primary key: (empID, projID)
Candidate key: (empID, projID)
Highest normal form: 3NF

e) Evaluation(projID*, evalDate, manager, rating)

Primary key: (projID, evalDate)
Candidate key: (projID, evalDate), (manager,evalDate)
Highest normal form: 1NF
Reason: Because 'manager' does not depend on the whole key (projID, evalDate) then it not meet the condition for 2NF

Question 3:
a) Department(deptID, deptName, manager, empID*)

Department_info (deptID, deptName, manager)
Department_member(empID, deptID*)

b) Employee(empID, empName, deptID*, email)

Employee_detailed(empID, empName, deptID)
Employee_contact(empID, email)

c) Evaluation(projID*, evalDate, manager, rating)

Project_score(projID, evalDate, rating)
Project_manager(manager, projID*)

Question 4:
After **decomposition**, we have the following relations:

Department_info (deptID, deptName, manager)
Department_member(empID, deptID*)
Employee_detailed(empID, empName, deptID)
Employee_contact(empID, email)
Project(projID, startYear, deptID*)
EmpProj(empID*, projID*, role)
Project_score(projID, evalDate, rating)
Project_manager(projID*, manager)

The resulting **final relational database** schema is:

Department_info (deptID, deptName, manager)
Employee_detailed(empID, empName, deptID)
Employee_contact(empID, email)
Project(projID, startYear, deptID*,manager)
EmpProj(empID, projID, role)
Project_score(projID, evalDate, rating)

The Department_member table is merged with Employee_detailed which converted to Employee_detailed table. This table is in 3NF as every attribute is only functionally dependent in primary key

The Project_manager table is merged with Project table, in this case, the project table is not in 3NF as this table has following dependencies:

Project(projID, startYear, deptID*,manager)
projID → startYear, deptID, manager
deptID → manager

Then the table is decomposed as two tables:

Project(projID, startYear, deptID*)
Project_manager(manager, projID*)

Then the final list of table is represented:

Department_info (deptID, deptName, manager)
Employee_detailed(empID, empName, deptID)
Employee_contact(empID, email)
Project(projID, startYear, deptID*)
Project_manager(manager, projID*)
EmpProj(empID, projID, role)
Project_score(projID, evalDate, rating)

PART B:

Question 1:

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

Question 2:

```
SELECT subjectid, COUNT (subjectid) AS total_books
FROM book
WHERE subjectid IS NOT NULL
GROUP BY subjectid
ORDER BY total_books DESC;
```

Question 3:

a)

```
SELECT DISTINCT person.firstname, person.lastname, person.city
FROM person, borrow
```

WHERE person.personid = borrow.personid;

b)

SELECT DISTINCT person.firstname, person.lastname, person.city
FROM person JOIN borrow
WHERE person.personid = borrow.personid;

c)

SELECT firstname, lastname, city
FROM person
WHERE personid **IN** (**SELECT DISTINCT** personid
FROM borrow);

Question 4:

```
SELECT book.bookdescid, book.title
FROM book
JOIN subject ON book.SUBJECTID = subject.SUBJECTID
JOIN written_by ON book.BOOKDESCID = written_by.BOOKDESCID
WHERE LOWER(subject.SUBJECTTYPE) LIKE 'databases'
GROUP BY book.BOOKDESCID
HAVING COUNT(DISTINCT written_by.AUTHORID) > 2
```

Question 5:

```
SELECT person.firstname || ' ' || person.lastname AS borrower_name,
        book.title, date(borrow.returndate), date(borrow.duedate), borrow.returndate -
borrow.duedate as late
From borrow JOIN person ON borrow.personid = person.personID
            JOIN borrow_copy ON borrow.transactionID = borrow_copy.transactionid
            JOIN book_copy on borrow_copy.bookid= book_copy.bookid
            JOIN book on book_copy.bookdescid = book.bookdescid
WHERE late > 0;
```

Question 6:

```
SELECT b.bookdescID, b.title, b.year
FROM book_copy AS bc LEFT JOIN borrow_copy AS br ON bc.bookID = br.bookID
JOIN book as b ON b.bookdescID = bc. bookdescID
WHERE transactionID IS NULL AND bc.bookID = b.bookdescID
GROUP BY bc.bookID;
```

Question 7:

```
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, book.title;
```

QUESTION 8:

```
SELECT book.title
FROM book JOIN written_by ON book.BOOKDESCID = written_by.BOOKDESCID
JOIN author ON written_by.AUTHORID = author.AUTHORID
WHERE LOWER(book.title) LIKE '%networks%'
AND (LOWER(author.firstname) = 'tim' AND LOWER(author.lastname) = 'miller'
OR LOWER(author.firstname) = 'jason' AND LOWER(author.lastname) = 'noel')
GROUP BY book.title
HAVING COUNT(DISTINCT author.authorid) = 2
```

QUESTION 9:

```
SELECT b.title, a.firstname || ' ' || a.lastname AS `Author Name`, b.year
FROM book b
JOIN written_by wb1 ON b.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 LOWER(b.title) = 'computer science' AND b.bookdescid != b2.bookdescid
```

Question 10:

```
SELECT person.firstname || ' ' || person.lastname AS "Borrower Name", book.TITLE,
subject.SUBJECTTYPE, date(borrow.BORROWDATE) AS "Borrow Date",
date(borrow.returndate) AS "Return Date"
FROM borrow join person on borrow.PERSONID = person.PERSONID
JOIN borrow_copy on borrow.TRANSACTIONID = borrow_copy.TRANSACTIONID
JOIN book_copy on borrow_copy.bookid = book_copy.bookid
JOIN book on book_copy.BOOKDESCID = book.BOOKDESCID
JOIN subject on subject.SUBJECTID = book.SUBJECTID
WHERE LOWER(subject.SUBJECTTYPE ) LIKE 'image processing'
```

PART C:

a)
The Dewey Decimal Classification (DDC) system implies the data integrity constraint in the database. As each book issued by a Dewey call number that describes its subject matter and class. This constrains ensure that each book is categorised exactly with the subject and class based on DDC number.

The given Library database schema does not enforce constraint in DDC, the “book” table has the “dewey” attributes and the “subject” table has “subjectID”. However, there is no constraint in these two tables based on the subject and class of the book. Therefore, the new foreign key in the “book” table referred to the subject in the “subject” table to ensure that each book subject is defined by its “dewey” number

b)
Without the relationship between the user and author, there is no direct way that we can find the author as a book user. Despite its little that the library user is the author, when it happens the relationship attributes (UserAsAuthor) can be introduced with the primary key from “Person” and “Author” table and other attributes. Other problem of lacking connection between Author and Person table is the complexity of query when retrieving data that required information in these two tables. For example, if the library wants to know which authors is most favourable based on the number of people borrowing their books. By introducing the relationship between Person and Author table it will reduce the step of joining tables. I recommend creates an associative table which includes “PersonID”, “AuthorID” and “BookID”.

c)
In the current database, “transaction” ID play a roles of recording the book borrowing activity from many people with different time. Moreover, multiple attributes is introduced along with “transactionID” such as borrow data, return date or mutate the new attributes such as total day late or penalty fee. Therefore, the “transactionID” in the given database is useful when retrieving data as it provides the accurate number book borrowed by people, which is no returned. With this information the library can easily find the person who kept the book, contact them and apply penalty on them if necessary. It will indicate the efficiency in managing books and reduce losses of library. Without the artificial “transactionID” it is more complicated to record the borrowing history. In order to uniquely identify a transaction, the composite of “PersonID”, “BookID”, “Borrow_date” is necessary to determine a transaction as a person can borrow several books at a time. And without the artificial key, there only one table record the borrowing activity and the specific number in each borrowing (there are two different table with artificial key). Then it is harder to know how many borrowing activity happens as the table does not separate the transaction by person but separate the transaction by books. In conclusion, using artificial is more efficient in this case due to it function as record history of borrowing activity.