Assessment 2: SQL Programming and Normalisation

PART A: Relational Database Design

1.

Department:

deptID : manager, empID, deptName

Employee:

empID : deptID, empName, email

email : deptID ,empID, empName

Project:

projID : startYear, deptID

EmpProj :

projID, role : empID

empID, projID: role

Evaluation:

projID, evalDate : rating

projID : manager

2.

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

Candidate keys: NONE

Primary key: deptID

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

It is not in the First normal form. The business rule says that a department can have multiple employees. Then, storing a single employee ID out of the possibility of multiple different employee IDs in the department table violates the first normal form.

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

Primary key : empID

Candidate key : email

Foreign key : deptID(Department.deptID)

It is 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, deptID*, startYear,)

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)

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 :- 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.

It is not in the second normal form as the ‘manager’ foreign key doesn’t depend on the ‘evalDate’ of the composite primary key. ‘Manager’ is determined based on the projID.

3.

The relations not in third normal form are

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

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

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

empID is an unnecessary foreign key in the department table.The business rule says that one department can have many employees. Then, storing a single employee ID out of a possibility of multiple different employee IDs in the department table contradicts the first normal form. We delete this foreign key attribute. There is no data loss as the employee table has the deptID as the foreign key. The department schema in the third normal form is

Department(deptID, deptName, manager*)

DepartmentEmployee(deptID*, empID*)

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

manager* is not necessary in foreign key in the Evaluation table and it doesn't depend on the whole primary key. We can delete 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 is

Evaluation(projID*, evalDate, rating)

ProjectED, deptName, 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*)

As we can see the relation defined by the schema, DepartmentEmployee(deptID, empID*) is also defined in the schema Employee(empID, empName, deptID*, email), we can combine these two.

The same goes for ProjectEvaluator(projID*, manager*). The relation between the projID and manager is also available through transitive functional dependency given by

projID -> deptID -> manager

Thus we can delete the ProjectEvaluator(projID*, manager*) schema as 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(empID*, projID*, role)

Evaluation(projID*, evalDate, rating)

Part B: SQL

1. SELECT c1.firstname || ' ' || c1.lastname AS 'Name',
c1.address || ', ' || c1.city AS 'Address'
FROM person c1
WHERE LOWER(c1.city) = 'portland';

2. SELECT subject.subjectID, COUNT(book_copy.bookID) as total_books
FROM book_copy
INNER JOIN book ON book_copy.bookdescID = book.bookdescID
INNER JOIN subject ON book.subjectID = subject.subjectID
GROUP BY subject.subjectID
ORDER BY total_books DESC;

3.a. SELECT person.firstname, person.lastname, person.city
FROM person, borrow
WHERE person.personID = borrow.personID
GROUP BY person.personID;

3.b. SELECT person.firstname, person.lastname, person.city
FROM person
JOIN borrow ON person.personID = borrow.personID
GROUP BY person.personID;

3.c. SELECT firstname, lastname, city
FROM person
WHERE personID IN (
SELECT personID
FROM borrow
);

4. SELECT book.bookdescID, book.title
FROM book

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

5.
```
SELECT book.title as "Book Title",
person.firstname || ' ' || person.lastname as "Borrower Name",
date(borrow.returndate) as "Return Date",
date(borrow.duedate) as "Due Date",
julianday(borrow.returndate) - julianday(borrow.duedate) as "Days Delayed"
FROM borrow_copy
JOIN borrow ON borrow_copy.transactionID = borrow.transactionID
JOIN book_copy ON borrow_copy.bookID = book_copy.bookID
JOIN book ON book_copy.bookdescID = book.bookdescID
JOIN person ON borrow.personID = person.personID
WHERE borrow.returndate > borrow.duedate
ORDER BY "Book Title" ASC, "Borrower Name" ASC, "Days Delayed" DESC;
```

6.
```
SELECT bookdescID, title, year
FROM book
WHERE bookdescID NOT IN
(SELECT bookdescID FROM book_copy WHERE bookID IN (SELECT bookID FROM
borrow_copy))
ORDER BY title ASC, year DESC;
```

7.
```
SELECT
author.Firstname, author.Lastname
,written_by.Role, book.Title
FROM
author, written_by, book
WHERE
author.authorID = written_by.authorID
AND written_by.bookdescID = book.bookdescID
AND written_by.role = 'Author'
AND author.authorID IN (
SELECT authorID
FROM written_by
WHERE role = 'Author'
GROUP BY authorID
HAVING COUNT( bookdescID) > 1
)
ORDER BY
author.lastname ASC,
author.firstname ASC;
```

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 '%network%'
AND written_by.authorID IN (SELECT authorID FROM author WHERE firstname = 'Tim'
AND lastname = 'Miller')
AND written_by.authorID IN (SELECT authorID FROM author WHERE firstname = 'Jason'
AND lastname = 'Noel')
AND NOT EXISTS (SELECT bookdescID FROM written_by WHERE bookdescID =
book.bookdescID AND authorID NOT IN
(SELECT authorID FROM author WHERE firstname = 'Tim' AND lastname = 'Miller' OR
firstname = 'Jason' AND lastname = 'Noel')) ;
```

9.
```
SELECT  book.title, author.firstname || ' ' || author.lastname AS "Author Name", book.year
FROM book
JOIN written_by ON book.bookdescID = written_by.bookdescID
JOIN author ON written_by.authorID = author.authorID
WHERE author.authorID IN (SELECT authorID FROM written_by WHERE bookdescID IN
(SELECT bookdescID FROM book WHERE LOWER(title) = 'computer science'))
AND book.bookdescID NOT IN
(SELECT bookdescID FROM book WHERE LOWER(title) = 'computer science')
ORDER BY "Author Name", book.year DESC;
```

10.
```
SELECT  person.firstname || ' ' || person.lastname AS "Borrower", book.title,
subject.subjecttype AS "Book Subject",
date(borrow.borrowdate) AS "Borrow Date", date(borrow.returndate) AS "Return Date"
FROM borrow
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 book.subjectID = subject.subjectID
JOIN person ON borrow.personID = person.personID
WHERE subject.subjecttype LIKE '%Image Processing%'
ORDER BY "Borrower", "Book Subject", "Borrow Date";
```

Part c: Research questions

1.The Dewey Decimal categorisation (DDC) system may be thought of as a hierarchical categorisation scheme that places restrictions on the Dewey call numbers and the subject classes of books in terms of data integrity. Dewey call numbers, for instance, must begin with a three-digit number that denotes the primary topic class, followed by a dot, and then one or more digits that denote a particular subclass. For example The subject class "Computer Science, Information, and General Works" and the subcategory "Computer programming, programmes, and data" are therefore required for a book with the Dewey call number 005.789.The foreign key 'dewey' in the book relation, which refers to the subjectID in the subject relation, is how the provided Library database structure implements this requirement. The restriction on the Dewey call number format is not, however, officially enforced.We may use a check constraint in the database schema to enforce this restriction. For the Dewey call numbers, we can define a check constraint that makes sure they are formatted correctly. For instance, a check constraint can be defined on the 'dewey' attribute of the book relation to make sure it matches the pattern '###.#' where '#' denotes a digit.

2. Because it does not account for the relationship between the person who borrows a book and the author, the current Library database ER model has this flaw. This may restrict the database's capacity to offer information on library customers' reading choices and borrowing habits. We may add a new relationship between person and author entities to enhance the ER model and the accompanying relational database structure. For instance, we may add a new connection called "interest" that links people to the authors or topics that interest them. Foreign keys referencing the person and author relations are both possible for this relation.

3. The "borrow" entity type and the "borrow" table both employ an artificial primary key called "transactionID" in the current Library database structure. Since the borrow table keeps track of the check-ins and check-outs of the books, including the person who borrows the book, the due date, and the return date, the real-world scenario of "users borrow books" is effectively modelled in the current schema.

However, it would be difficult to keep track of multiple books borrowed by the same person in a single transaction if we were to remove the fictitious primary key "transactionID". Let's say someone takes out three books from the library. In that instance, we would need to create a new table, whose primary key would be a combination of the borrowdate and personID, to link several books to a single transaction. Artificial primary keys can therefore be used to increase data integrity and simplify the data model, but it is crucial to watch out that they do not cause data fragmentation.