Homework 5

Name: Dejun Qian
PSUID: 917233450

Question 1:

Create two relations where student is identified by id and advisor is identified by id:

Student (id, first, last, advisor_id)

Advisor (id, first, last, student_id)

Each student must have an advisor and each advisor must have a student assistance – identified by student-id. Define the required foreign key and not null constraints.

Demonstrate that you cannot add a new student with a new advisor in the student table. Also, demonstrate that you cannot add a new advisor with a new student in the advisor table. (You also can't add a student with a null advisor or an advisor with a null student assistant. We'll learn about how to overcome these problems in about two weeks.)

Provide SQL statements and screen shots.

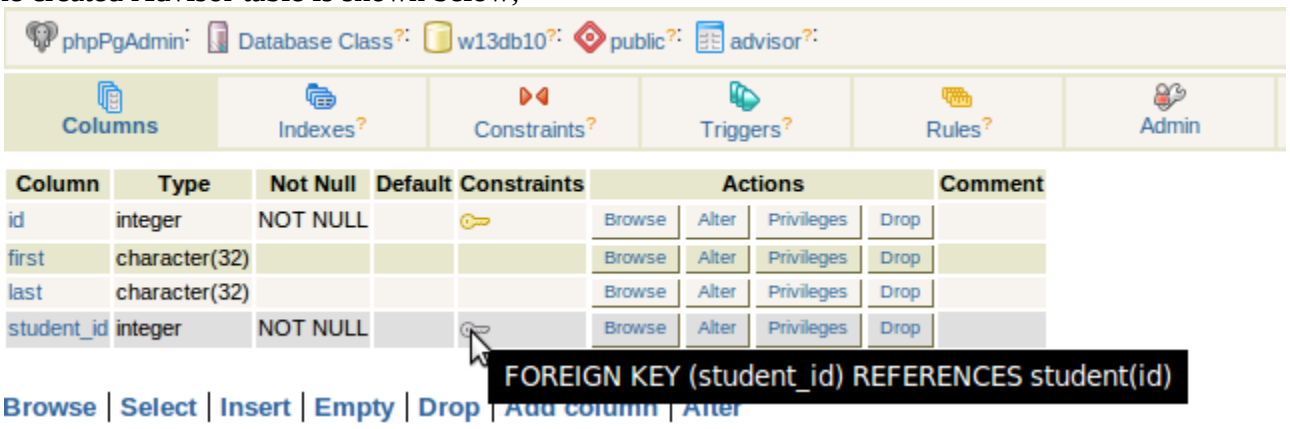
Answer:

The following is the SQL statements for creating the relations,

```
CREATE TABLE Student (
id INTEGER PRIMARY KEY,
first CHAR(32),
last CHAR(32),
advisor_id INTEGER NOT NULL
)
CREATE TABLE Advisor (
id INTEGER PRIMARY KEY,
first CHAR(32),
last CHAR(32),
student_id INTEGER NOT NULL REFERENCES Student(id)
)
ALTER TABLE Student ADD CONSTRAINT c1 FOREIGN KEY (advisor_id) REFERENCES Advisor(id)
```

The created Student table is shown below,

The created Advisor table is shown below,



After we created the tables, both of the tables are empty. We begin the following experiment based on these empty tables.

The following is the SQL statements to demonstrate that we cannot add a new student with a new advisor in the student table,

```
INSERT INTO Student VALUES (1,'Tom','Gates',1)
```

The result is shown below,

Query Results

SQL error:

```
ERROR:  insert or update on table "student" violates foreign key constraint "c1"
DETAIL:  Key (advisor_id)=(1) is not present in table "advisor".
```

In statement:
INSERT INTO Student VALUES (1,'Tom','Gates',1)

The following is the SQL statements to demonstrate that we cannot add a student with a null advisor in the student table,

```
INSERT INTO Student VALUES (1,'Tom','Gates',NULL)
```

The result is shown below,

Query Results

SQL error:

```
ERROR:  null value in column "advisor_id" violates not-null constraint
```

In statement:
INSERT INTO Student VALUES (1,'Tom','Gates',NULL)

The following is the SQL statements to demonstrate that we cannot add a new advisor with a new student in the advisor table,

```
INSERT INTO Advisor VALUES (1,'Tom','Gates',1)
```

The result is shown below,

Query Results

SQL error:

```
ERROR:  insert or update on table "advisor" violates foreign key constraint "advisor_student_id_fkey"
DETAIL:  Key (student_id)=(1) is not present in table "student".
```

In statement:
INSERT INTO Advisor VALUES (1,'Tom','Gates',1)

The following is the SQL statements to demonstrate that we cannot add a advisor with a null student in the advisor table,

```
INSERT INTO Advisor VALUES (1,'Tom','Gates',NULL)
```

The result is shown below,

Query Results

SQL error:

```
ERROR:  null value in column "student_id" violates not-null constraint
```

In statement:
INSERT INTO Advisor VALUES (1,'Tom','Gates',NULL)

Question 2:

Consider the following relations:

Title (ISBN, title, author, yearPublished, publisher)
Copy (ISBN, copyNum, yearPurchased, onLoan)
Member (id, first, last, age, gender)
Loan (memberId, ISBN, copyNum, outDate, dueDate)

The Title relation has one row for each book in the library listed, with its ISBN, title, year it is published, and publisher. A book can have many physical copies. Each copy has a copy number and is listed in the Copy relation. A member of this library has a unique ID and is able to check out a copy of a book, represented by a row in the Loan table.

a. Write SQL statements to create the tables with appropriate primary key and foreign key constraints. Give screen shots of the results.

b. Write tuple or attribute level CHECK constraints or Assertions to ensure that each of the following requirement is met:

i. All the members should be at least 10 years old and should have a valid gender (of 'm' or 'f').

ii. The year purchased for a book copy should be greater than or equal to the year the book was published.

iii. The total copies of books loaned to a member below age 18 should not exceed 5.

c. Write a trigger for the following requirement and implement it in the PostgreSQL:

Whenever a members loans a copy of a book, it should be marked as onLoan = 'Y'. Provide your code, screen shots for your implementation and screen shots for the execution of the trigger. You will have to add the required data in your database.

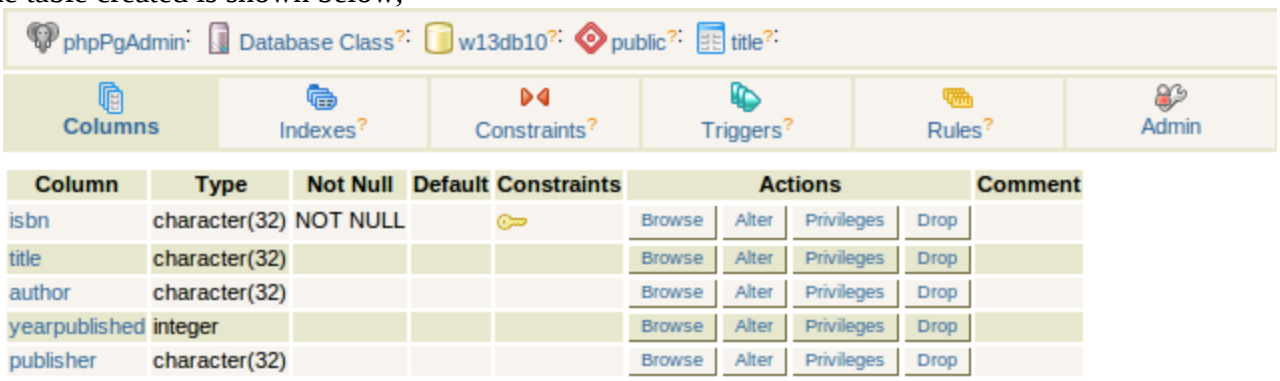
Answer:

a.

The following is the SQL statement to create Title table,

```
CREATE TABLE Title (
ISBN CHAR(32) PRIMARY KEY,
title CHAR(32),
author CHAR(32),
yearPublished INTEGER,
publisher CHAR(32)
)
```

The table created is shown below,



Column	Type	Not Null	Default	Constraints	Actions	Comment
isbn	character(32)	NOT NULL			Browse Alter Privileges Drop	
title	character(32)				Browse Alter Privileges Drop	
author	character(32)				Browse Alter Privileges Drop	
yearpublished	integer				Browse Alter Privileges Drop	
publisher	character(32)				Browse Alter Privileges Drop	

The following is the SQL statement to create Copy table,

```
CREATE TABLE Copy (
ISBN CHAR(32) REFERENCES Title(ISBN),
copyNum INTEGER,
yearPurchased INTEGER,
onLoan CHAR,
PRIMARY KEY (ISBN,copyNum)
)
```

The table created is shown below,

Column	Type	Not Null	Default	Constraints	Actions	Comment
isbn	character(32)	NOT NULL			Browse Alter Privileges Drop	
copynum	integer	NOT NULL				
yearpurchased	integer					
onloan	character(1)				Browse Alter Privileges Drop	

The following is the SQL statement to create Member table,

```
CREATE TABLE Member (
id INTEGER PRIMARY KEY,
first CHAR(32),
last CHAR(32),
age INTEGER,
gender CHAR
)
```

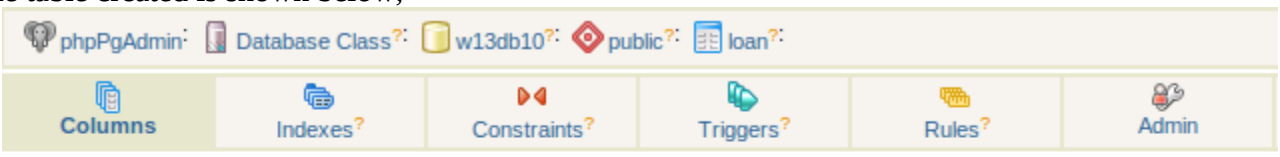
The table created is shown below,

Column	Type	Not Null	Default	Constraints	Actions	Comment
id	integer	NOT NULL			Browse Alter Privileges Drop	
first	character(32)				Browse Alter Privileges Drop	
last	character(32)				Browse Alter Privileges Drop	
age	character(32)				Browse Alter Privileges Drop	
gender	character(1)				Browse Alter Privileges Drop	

The following is the SQL statement to create Loan table,

```
CREATE TABLE Loan (
memberId INTEGER,
ISBN CHAR(32),
copyNum INTEGER,
outDate DATE,
dueDate DATE,
PRIMARY KEY(memberId,ISBN,copyNum),
FOREIGN KEY (memberId) REFERENCES Member(id),
FOREIGN KEY (ISBN,copyNum) REFERENCES Copy(ISBN,copyNum)
)
```

The table created is shown below,



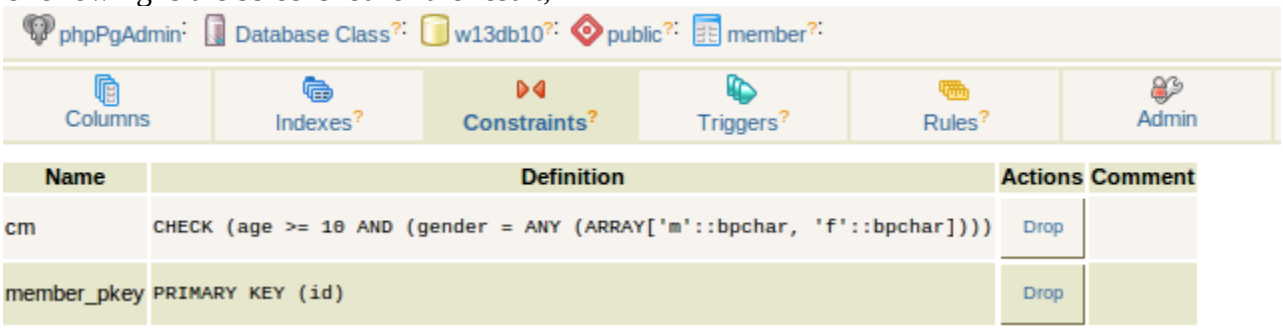
Column	Type	Not Null	Default	Constraints	Actions	Comment
memberid	integer	NOT NULL			Browse Alter Privileges Drop	
isbn	character(32)	NOT NULL			Browse Alter Privileges Drop	
copynum	integer	NOT NULL			Browse Alter Privileges Drop	
outdate	date				Browse Alter Privileges Drop	
duedate	date				Browse Alter Privileges Drop	

b.

i. The following is the SQL statement to create the required constraint,

```
ALTER TABLE Member ADD CONSTRAINT cm CHECK(age>=10 AND gender IN ('m','f'))
```

The following is the screenshot for the result,



Name	Definition	Actions	Comment
cm	CHECK (age >= 10 AND (gender = ANY (ARRAY['m'::bpchar, 'f'::bpchar])))	Drop	
member_pkey	PRIMARY KEY (id)	Drop	

ii. The following is the SQL statement to create the required constraint,

```
CREATE ASSERTION ac CHECK(Copy.yearPurchased >= (SELECT yearPublished FROM Title WHERE
ISBN=Copy.ISBN))
```

iii. The following is the SQL statement to create the required constraint,

```
CREATE ASSERTION am CHECK(
  NOT EXISTS(
    (SELECT memberId FROM Loan GROUP BY memberId HAVING COUNT(*)>5)
     INTERSECT
    (SELECT id FROM Member WHERE age<18)
  )
)
```

c.

The code for the trigger is shown below,

```
CREATE FUNCTION loanTrigger ()
RETURNS trigger AS
$$
BEGIN
UPDATE Copy SET onLoan = 'Y' WHERE ISBN=New.ISBN AND copyNum=New.copyNum;
RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER addLoanTrigger
AFTER INSERT ON Loan
FOR EACH ROW EXECUTE PROCEDURE loanTrigger();
```

To test the trigger, I issue the following SQL statements to add the initial data.

```
INSERT INTO Title VALUES ('11-1111','picture','Tom',1982,'publisher01'),
                         ('22-2222','animal','Helen',2004,'publisher02');
INSERT INTO Copy VALUES ('11-1111',1,1983,'N'),
                        ('11-1111',2,1984,'N');
INSERT INTO Member VALUES (1,'Eric','Starr',13,'m'),
                          (2,'David','Black',17,'m');
```

The following SQL statement add a loan to the table.

```
INSERT INTO Loan VALUES (1,'11-1111',1,DATE'1987-02-14',DATE'1987-08-14');
```

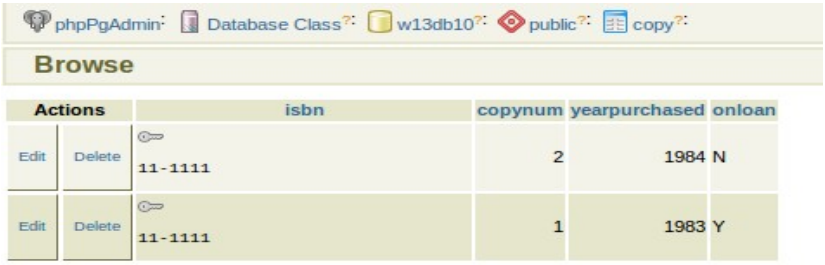
The result is shown bellow.
This is the Loan table.


phpPgAdmin: Database Class: w13db10: public: loan:

Browse

Actions		memberid	isbn	copynum	outdate	duedate
Edit	Delete	1	11-1111	1	1987-02-14	1987-08-14

This is the Copy table.


phpPgAdmin: Database Class: w13db10: public: copy:

Browse

Actions		isbn	copynum	yearpurchased	onloan
Edit	Delete	11-1111	2	1984	N
Edit	Delete	11-1111	1	1983	Y

Question 3:
Write a stored procedure for correcting clearance IDs of agents assigned to a particular mission (similar to Assignment 2).
The stored procedure should take a mission ID as an input. It should then check the access_id required for that mission and make sure that all the agents assigned to the mission have a security clearance that is less than or equal to the mission clearance. If not, update the agent's clearance to the required level. For example, if the mission clearance (i.e., access_id) is 3 and the agent's clearance_id is 4, then the agent's clearance_id should be changed to 3. If agent's clearance_id is 2, it should not be changed. The stored procedure should return the agents whose clearance ID was modified in the form of a cursor/ table with the following columns: AgentId, FirstName, LastName, OldClearanceID, NewClearanceID. You can either write one stored procedure for the entire functionality or you can write more than one stored procedures where one procedure invokes another (making it like an application) or you can write a stored procedure(s) and invoke it(them) from your C/ Java program in Assignment 2.

Answer:
The statements used are shown below. The main code is in code.txt. Instructions are shown in instruction.txt. The screenshots of the output are shown in mission1.png and mission5.png. The csv files are in mission1.csv and mission5.csv.

```
CREATE TYPE RESTYPE AS (agentid INTEGER, firstname CHAR(32), lastname CHAR(32), oldclearanceid INTEGER, newclearanceid INTEGER);

CREATE OR REPLACE FUNCTION update_clearance(missionid INTEGER)
RETURNS SETOF RESTYPE
AS $$
```

```
DECLARE
  resrow RESTYPE;
  row RECORD;
BEGIN
  FOR row IN SELECT agent.agent_id,agent.first,agent.last,agent.clearance_id,mission.access_id FROM
mission,teamrel,agent WHERE mission.mission_id=missionid AND mission.team_id=teamrel.team_id AND
teamrel.agent_id=agent.agent_id
    LOOP
      IF (row.access_id < row.clearance_id) THEN
        resrow.agentid := row.agent_id;
        resrow.firstname := row.first;
        resrow.lastname := row.last;
        resrow.oldclearanceid := row.clearance_id;
        resrow.newclearanceid := row.access_id;
        UPDATE agent SET clearance_id=row.access_id WHERE agent_id=row.agent_id;
        RETURN NEXT resrow;
      END IF;
    END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM update_clearance(1)
SELECT * FROM update_clearance(5)

DROP FUNCTION update_clearance
DROP TYPE RESTYPE
```

The result of mission id 1 is also pasted here.

Query Results

No rows found.

Total runtime: 8.862 ms

SQL executed.

Here is the result for mission id 5.

Query Results

agentid	firstname	lastname	oldclearanceid	newclearanceid
258	John	Miller	5	3
353	Tim	McCracken	5	3
498	Tim	Divola	5	3
812	Kate	Hall	5	3
920	Tamara	Lonborg	5	3

5 row(s)

Total runtime: 13.220 ms