Lab 6 Amirlan Sharipov (BS-CS21-01)

Amirlan Sharipov (BS-CS21-01)

February 28, 2023

Contents

1 Disclaimer

Please, use the lab6.sql file to read/copy the source code. Also, the html version of this document looks better than the pdf one.

2 Exercise 1

2.1 Table creation and insertion

I used the schema provided in the slides. And then manually inserted data into the tables.

```
CREATE TABLE customers (
    customerId INT,
    customerName VARCHAR(50),
    city VARCHAR(50),
    PRIMARY KEY (customerId)
);

CREATE TABLE items (
    itemId INT,
    itemName VARCHAR(50),
    price FLOAT,
    PRIMARY KEY (itemId)
);

CREATE TABLE orders (
    orderId INT,
    customerId INT,
    date DATE,
    FOREIGN KEY (customerId) REFERENCES customers(customerId),
    PRIMARY KEY (orderId)
);

CREATE TABLE order_details (
    orderId INT,
    itemId INT,
    quantity INT,
    FOREIGN KEY (itemId) REFERENCES items(itemId),
    PRIMARY KEY (orderId, itemId)
);

INSERT INTO customers VALUES('101', 'Martin', 'Prague');
INSERT INTO customers VALUES('107', 'Herman', 'Madrid');
INSERT INTO customers VALUES('110', 'Pedro', 'Moscow');

INSERT INTO items VALUES('3786', 'Net', 35.0);
INSERT INTO items VALUES('4011', 'Racket', 65.0);
INSERT INTO items VALUES('9132', 'Pack-3', 4.75);
INSERT INTO items VALUES('5794', 'Pack-6', 5.0);
INSERT INTO items VALUES('3141', 'Cover', 10.0);
```

```
INSERT INTO orders VALUES('2301', '101', '2011-02-23');
INSERT INTO orders VALUES('2302', '107', '2011-02-25');
INSERT INTO orders VALUES('2303', '110', '2011-02-27');

INSERT INTO order_details VALUES ('2301', '3786', 3);
INSERT INTO order_details VALUES ('2301', '4011', 6);
INSERT INTO order_details VALUES ('2301', '9132', 8);
INSERT INTO order_details VALUES ('2302', '5794', 4);
INSERT INTO order_details VALUES ('2303', '4011', 2);
INSERT INTO order_details VALUES ('2303', '3141', 2);
```

2.2 Queries

First query takes the sum of all prices * quantities groupped by the orders and sorts them. The second query does the same thing, groups by customers, sorts by sum of quantities (descending order) and takes the first result.

```
SELECT order_details.orderId, SUM(items.price * order_details.quantity)
FROM order_details
INNER JOIN items on items.itemId=order_details.itemId
group by order_details.orderId ORDER BY sum ASC;

SELECT customers.customerName, customers.city FROM customers
INNER JOIN orders ON customers.customerId=orders.customerId
INNER JOIN order_details ON order_details.orderId=orders.orderId
GROUP BY customers.customerId
ORDER BY SUM(order_details.quantity) DESC
LIMIT 1;
```

3 Exercise 2

3.1 Normalization

I have several assumptions:

- Any teacher can work at several schools at once (or change the school)
- Room numbers don't depend on schools: the first number of the room name is not enough to assume the opposite
- Teachers may teach several courses: it's an elementary school, usually teachers can teach anything in elementary schools

3.1.1 1NF

It's almost in 1NF state. Each cell is already atomic, values of the same domain, etc. The only thing that's not there is the primary key. Let's say for now that the primary key is a tuple of school, teacher, course, room, grade, and book. This way there are no conflicts. It still looks like a mess, so I will normalize it further.

3.1.2 2NF

Make new tables with relations for partial functional dependencies of non-prime attributes on candidate keys:

- lessons (lessonId, schoolName, teacherName, courseName, roomName, gradeName)
- books (bookId, bookName, publisherName)
- loans (loanId, lessonId, bookId, loanDate)

3.1.3 3NF

Make new tables (with appropriate IDs) with relations for transitive functional dependencies of non-prime attribute on candidate key:

- schools (schoolId, schoolName)
- teachers (teacherId, teacherName)
- courses (courseId, courseName)
- rooms (roomId, roomName)
- grades (gradeId, gradeName)
- publishers (publisherId, publisherName)
- lessons (lessonId, schoolId, teacherId, courseId, roomId, gradeId)
- books (bookId, bookName, publisherId)
- loans (loanId, lessonId, bookId, loanDate)

3.1.4 BCNF and 4NF

Already satisfies.

3.1.5 Code

```
CREATE TABLE schools (
    schoolId SERIAL,
    schoolName VARCHAR(50),
    PRIMARY KEY (schoolId)
);

CREATE TABLE teachers (
    teacherId SERIAL,
    teacherName VARCHAR(30),
    PRIMARY KEY (teacherId)
);

CREATE TABLE courses (
    courseId SERIAL,
    courseName VARCHAR(40),
    PRIMARY KEY (courseId)
);

CREATE TABLE rooms (
    roomId SERIAL,
    roomName VARCHAR(40),
    PRIMARY KEY (roomId)
);

CREATE TABLE grades (
    gradeId SERIAL,
    gradeName VARCHAR(15),
    PRIMARY KEY (gradeId)
);

CREATE TABLE publishers (
    publisherId SERIAL,
    publisherName VARCHAR(30),
    PRIMARY KEY (publisherId)
);

CREATE TABLE books (
    bookId SERIAL,
```

```
    bookName VARCHAR(60),
    publisherId INT,
    FOREIGN KEY (publisherId) REFERENCES publishers(publisherId),
    PRIMARY KEY (bookId)
);

CREATE TABLE lessons (
    lessonId SERIAL,
    schoolId INT,
    teacherId INT,
    courseId INT,
    roomId INT,
    gradeId INT,
    FOREIGN KEY (teacherId) REFERENCES teachers(teacherId),
    FOREIGN KEY (courseId) REFERENCES courses(courseId),
    FOREIGN KEY (roomId) REFERENCES rooms(roomId),
    FOREIGN KEY (gradeId) REFERENCES grades(gradeId),
    PRIMARY KEY (lessonId)
);

CREATE TABLE loans (
    loanId SERIAL,
    lessonId INT,
    bookId INT,
    loanDate DATE,
    FOREIGN KEY (lessonId) REFERENCES lessons(lessonId),
    FOREIGN KEY (bookId) REFERENCES books(bookId),
    PRIMARY KEY (loanId)
);

INSERT INTO schools (schoolName)
    SELECT DISTINCT school FROM loan_books;
INSERT INTO teachers (teacherName)
    SELECT DISTINCT teacher FROM loan_books;
-- Inserted Numerical thinking 2 times because of case sensitivity. Not gonna change ar
INSERT INTO courses (courseName)
    SELECT DISTINCT course FROM loan_books;
INSERT INTO rooms (roomName)
    SELECT DISTINCT room FROM loan_books;
INSERT INTO grades (gradeName)
```

```
    SELECT DISTINCT grade FROM loan_books;
INSERT INTO publishers (publisherName)
    SELECT DISTINCT publisher FROM loan_books;
INSERT INTO books (bookName, publisherId)
    SELECT DISTINCT loan_books.book,
    publishers.publisherId FROM loan_books
    INNER JOIN publishers ON
        publishers.publisherName=loan_books.publisher;

INSERT INTO lessons (schoolId, teacherId, courseId, roomId, gradeId)
    SELECT DISTINCT schools.schoolId, teachers.teacherId,
        courses.courseId, rooms.roomId, grades.gradeId FROM loan_books
    INNER JOIN schools ON schools.schoolName=loan_books.school
    INNER JOIN teachers ON teachers.teacherName=loan_books.teacher
    INNER JOIN courses ON courses.courseName=loan_books.course
    INNER JOIN rooms ON rooms.roomName=loan_books.room
    INNER JOIN grades ON grades.gradeName=loan_books.grade;

INSERT INTO loans (lessonId, bookId, loanDate)
    SELECT DISTINCT lessons.lessonId, books.bookId,
        loan_books.loanDate FROM loan_books
    INNER JOIN schools ON schools.schoolName=loan_books.school
    INNER JOIN teachers ON teachers.teacherName=loan_books.teacher
    INNER JOIN courses ON courses.courseName=loan_books.course
    INNER JOIN rooms ON rooms.roomName=loan_books.room
    INNER JOIN grades ON grades.gradeName=loan_books.grade
    INNER JOIN books ON books.bookName=loan_books.book
    INNER JOIN lessons ON lessons.gradeId=grades.gradeId
        and lessons.roomId=rooms.roomId
        and lessons.courseId=courses.courseId
        and lessons.teacherId=teachers.teacherId;
```

3.2 Queries

The first query list all the schools that borrowed the books of every publisher using DISTINCT keyword. The second query orders the results from each school, and takes only 1 loan that has the highest loanDate for each school.

```
SELECT DISTINCT publishers.publisherName, books.bookName,
    schools.schoolName FROM books
```

```
    INNER JOIN loans ON loans.bookId=books.bookId
    INNER JOIN lessons ON lessons.lessonId=loans.lessonId
    INNER JOIN schools ON schools.schoolId=lessons.schoolId
    INNER JOIN publishers ON publishers.publisherId=books.publisherId
    ORDER BY publishers.publisherName;

SELECT DISTINCT ON (schools.schoolName) schools.schoolName,
    publishers.publisherName, books.bookName FROM loans
    INNER JOIN lessons ON lessons.lessonId=loans.lessonId
    INNER JOIN schools ON schools.schoolId=lessons.schoolId
    INNER JOIN books ON books.bookId=loans.bookId
    INNER JOIN publishers ON publishers.publisherId=books.publisherId
    ORDER BY schools.schoolName, loans.loanDate DESC, 1;
```