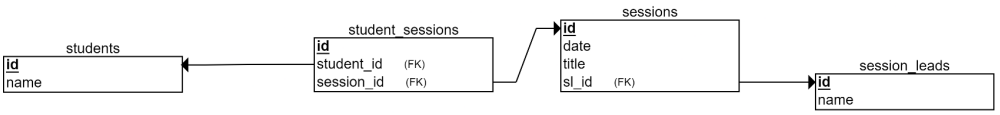
[Mohamed Raafat]

THEORETICAL

1. The tables and the fields that are needed to achieve the required functionality
2. The relationships between the tables

I finished my search



i think the current architecture is good from my point of view,
the students table and session leads are separate from each other allowing us to create many to many relationships, a session lead have many sessions to be precise 10 sessions during the span of 10 weeks, but the session have only one lead who is hossam so i will agree with that one to many relationship between the session_leads table and the session table, next is the many to many relationship between the students table and the sessions table, a student have 10 sessions during this course and the session have many students in it so creating a join table connecting those two tables seems good to me as they can't have a many to many relationship without creating that join table that we can describe as intermediary between them, but what seems redundant and resource intensive from my point of view is when the student need to know who is the session lead, this will need a lookup first with the student id in the student sessions table in order to get that session id then you will need to use that session id to lookup the student lead id in the sessions table then use that session lead id to look up the name of the session lead in the session lead table, a simple solution i guess will be including sl_id in the students table which references the id in the session_leads table, that way the above figure comes to full circle, if a student need to know his lead he will do a single lookup with the sl_id in the session_leads table, however this connection to make fast lookups for session lead may come at a cost of expendability or simplicity, at the end of the day it is just tradeoffs

---

[Doaa Ashraf Taha]

PRACTICAL

1. Create a migration file with the required SQL statements for creating the tables.
2. Create the needed SQL statements for the CRUD operations

I finished my search

How to make migrations?

1. Install the required dependencies

    - npm i -g db-migrate (installing it globally will allow you to use its terminal commands)
    - npm i db-migrate db-migrate-pg

2. to get migrations into live we use this terminal command

    - db-migrate create [tableName-table] --sql-file
    - for ex: lets suppose we have a table called session_leads
    - the line command for creating a migration for this table would be like this
        - db-migrate create session-leads-table --sql-file
        - after running this command a migration folder will be created with a sql folder inside it
        - the sql folder will have 2 files one for up migration the other for down migration
        - in the up file we add the sql command that create the session_leads table

            CREATE TABLE IF NOT EXISTS session_leads(id SERIAL PRIMARY KEY, name VARCHAR(50) NOT NULL);

        - in the down file add this sql command

            DROP TABLE IF EXISTS session_leads;

        - the very last step is to run the migration with this line command

            db-migrate up (to create the table **)**

            -or-

            db-migrate down (to remove the table)

- **important things to note**:
    - don't forget the semicolon in the up and down sql files

- ❖ don't forget to add the database.json file in the root of the project with the required code

CRUD Queries

- ★ **CREATE** INSERT INTO session_leads (name) VALUES ('Hossam Abu-Bakr'); this will add a new row in session_leads table.
- ★ **READ** SELECT* FROM session_leads; this will display all rows in the session_leads table.
- ★ **UPDATE** UPDATE session_leads SET name= 'Ahmed' WHERE id='1'; this will change the information in a specific row .
- ★ **DELETE** DELETE FROM session_leads WHERE id='1'; this will delete a specific row from the session_leads table.

[Mohamed Aboarab]

PRACTICAL

1. Create a migration file with the required SQL statements for creating the tables.
2. Create the needed SQL statements for the CRUD operations

I finished my search

1. follow up the following steps:
    a. install pg ,db-migrate and db-migrate-pg:
    ```npm i pg```
    ```npm i -D db-migrate db-migrate-pg @types/pg```
    b. create the migration file :
    ```npx db-migrate create add-tableName-table –sql-file
    c. in the created Up file,insert:
    ```CREATE TABLE tableName (firstColumn type,secondColumn type,..)
    d. in the created Down file,insert:
    ```DROP TABLE tableName```
2. the CRUD SQL statements:
    a. Create:
    ```INSERT INTO tableName (firstColumn,secondColumn,...)
    VALUES(val1,val2,....)```
    b. Read:

```SELECT * FROM tableName WHERE  conditions``` OR
```SELECT firstColumn,secondColumn FROM tableName WHERE conditions```

c. Update:
```UPDATE tableName SET ColumnName=value WHERE conditions```

d. delete:
```DELETE FROM tableName WHERE columnName=value```

[Mohamed Raafat]

# PRACTICAL

1. Create a migration file with the required SQL statements for creating the tables.
2. Create the needed SQL statements for the CRUD operations

I finished my search

**First we need to install the db-migrate package and it's driver in ordre to work with postgres as a development dependency**

```
npm i -D db-migrate db-migrate-pg
```

*** now we need to run the db-migrate module in order to create our first migration after creating the database.json file so that db-migrate-pg could work**

```
db-migrate create create-session-leads –sql-file
```

this will create 2 sql files one ending with *up.sql which will create our table and the other ending with *down.sql which will delete what the up has done.

now we can create the the create table in the up file

```
CREATE TABLE session_leads(
    id SERIAL PRIMARY KEY,
    name VARCHCAR(50) NOT NULL
);
```

and the down file will contain the reverse instruction of the up

```
DROP TABLE session_leads;
```

* now to add the students table we need to create another migration with the command

```
db-migrate create create-students-table –sql-file
```

this command will create another two files the up and down

in the up we will create the students table and that table will have a foreign key that references the session_leads table

```
CREATE TABLE students(
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    sl_id INTEGER NOT NULL REFERENCES session_leads(id)
);
```

in the down file we will create the reverse command

```
DROP TABLE students;
```

now after reviewing that relationship we figured out that the foreign key needed to be removed so we will create another migration with the command

```
db-migrate create remove-foreign-key –sql-file
```

in the up file:

```
ALTER TABLE students DROP COLUMN sl_id;
```

in the down file we will create that column again, note that we needed to respecify that sl_id is a foreignkey that references the id in students table otherwise it will be a normal column without any relation to the students table or you may get an error for not specifying the type of the column:

```
ALTER TABLE students CREATE COLUMN sl_id INTEGER REFERENCES students(id);
```

now we will create another migration that will create the sessions table using the command

```
db-migrate create create-sessions-table –sql-file
```

in the up file:

```
CREATE TABLE sessions (
    id SERIAL PRIMARY KEY,
    date DATE,
    title VARCHAR(100),
    time TIME,
    sl_id REFERENCES session_leads(id)
```

```
);
```

in the DROP file:

```
DROP TABLE SESSIONS;
```

last but not least we will create the last migration -that will create the student_session table which will connect the students table with the sessions table by a relation of many to many - using the command

```
db-migrate create create-student-session-table –sql-file
```

in the up file:

```
CREATE TABLE student_session (
    id SERIAL PRIMARY KEY,
    session_id INTEGER REFERENCES   sessions(id),
    student_id INTEGER REFERENCES students(id)
);
```

in the down file:

```
DROP TABLE student_session;
```

run the command db-migrate up to update the database to the latest schema

CRUD operations on students;

create: INSERT INTO students name=’Raafat’;
read: SELECT * FROM students;
update: UPDATE students SET name=’calvin’ WHERE id=’1’;
delete: DELETE FROM students WHERE name= ‘Raafat’;

CRUD operation on session_leads;

create: INSERT INTO session_leads name=’Hossam’;
read: SELECT * FROM session_leads;
update: UPDATE session_leads SET name=’James’ WHERE id=’1’
delete: DELETE FROM session_leads WHERE name = ‘Hossam’

the student_session and session are join tables so they only support read operations;