Submitted To:

Prof. Julia Stoyanovich

Principles Of Databases

Project Part I – Database Design

Submitted By:

Prasant Adhikari

NetID: pa1038

PORT: 8601

Sonali Singh

NetID: sks9370

PORT: 8681

Github Repository: https://github.com/prasantadh/principles-of-db-f22

A Database Application with a Web Front-End

Project Description

Problem Statement: Our project's aim is to find possible substitute teachers among the teaching staff for a lesson when a teacher is absent as well as showcase several other schedules that can be used in a school. However, we want to find the teacher who is best suited for the lesson – someone who is available, has expertise on the subject, and hasn't been overloaded with substitute lessons during that term or with their own lessons on that particular day. We intend to approach the task as a matching problem between a teacher and a lesson in need of a teacher.

As an additional challenge, we will attempt to create a schedule for the school at the beginning of each term. The ER model is designed based on the data we will need to store in order to create such a schedule. The actual implementation of the schedule creation is contingent on us having the time. As a start, we will simply work with the schedule that is already in place at the school and attempt to find substitute teachers.

Data Concerns and Constraints: We model a school with its departments, grades, subjects, rooms, teachers, students and lessons. The real life application is obvious and we will capitalize on it by drawing real data from a real high school that one of the team members attended. As a proof of concept project for this class, we are likely to work with a subset of the school's teachers and schedule data and not the whole dataset.

Real life applications come with real life constraints. We have decided to take into consideration the constraints from the school that is providing us with the data to model the school environment and find a relevant substitute teacher.

Queries: The queries our users will be able to ask are mostly regarding the teachers and the substitute teachers. Our main queries include the following:

1. Who teaches who?
2. What do they teach?
3. If someone is absent, who do we have available for substitution
4. What does this available teacher have in their day and have in the way of knowing the subject they are substituting for?
5. Can we have multiple teachers absent and still have someone available to fill all of their lessons?

Model Description

Entity Sets:

We leave out Schools as an entity set even though we would like to scale our application to multiple schools eventually. This is to keep the design simple, as we will only be dealing with a single school for the class project. Listed below are the entity sets we will be using.

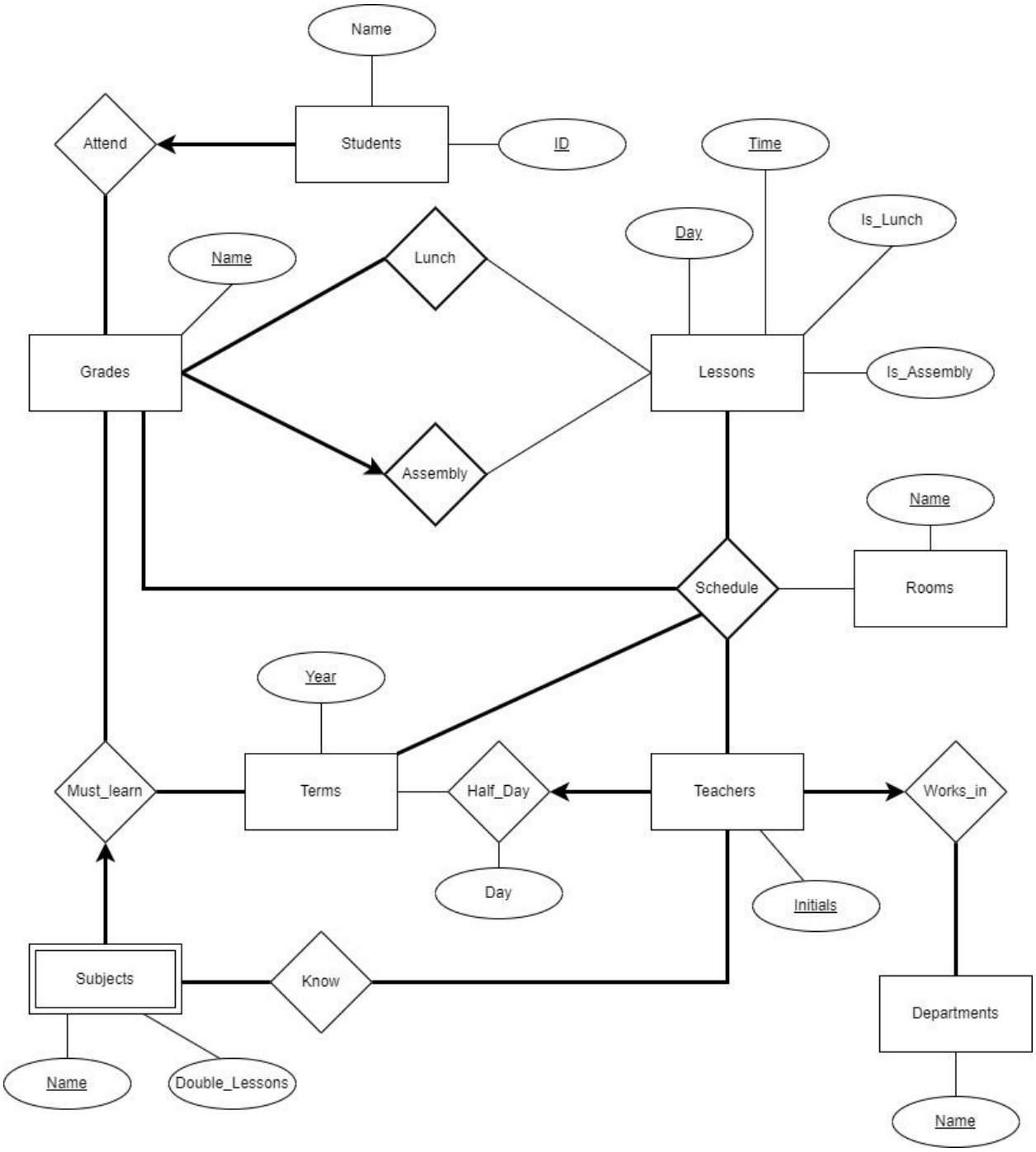
1. *Departments:* A School has multiple departments. Each Department is represented as an entity set identified by its name.
2. *Teachers:* A school has teachers. Each teacher is identified by their initials.
3. *Students:* A school has students. Each student is identified by their id number. While there could be more attributes that describe a student, we keep this minimal to what information we need from them to find them a substitute teacher.
4. *Grades:* Students are grouped into individual Grades identified by their names.
5. *Subjects:* Subjects are taught at a school. We model this with a Subjects entity set.
6. *Lessons:* Teachers and Grades meet at a specific day and time. We model this with a Lessons entity set.
7. *Rooms:* Each Lesson happens in a physical room. We model this with a Rooms entity set.
8. *Terms:* Schools design their schedules by Terms. We model this with a Terms entity set.

Business Rules

1. A Department is identified by its name. Each department has at least one Teacher.
2. A Teacher is identified by their initials.
    a. A Teacher is associated with exactly 1 department.
    b. A Teacher knows at least one Subject.
    c. A Teacher is entitled to have one day of the week assigned as their half day each term. On their half day, they don't take any lessons after the last lunch lesson. For simplicity of ER diagrams and data handling, we will only store the half day data for teachers for the current term.
    d. A Teacher may or may not have a specific Lesson scheduled but will have at least one Lesson scheduled.
    e. A Teacher cannot be scheduled for only multiple lessons that run on the same day and at the same time.
3. A Student is identified by their id and attends exactly 1 Grade.
4. A Grade is identified by its name.
    a. A Grade must learn at least one Subject.
    b. Each grade is assigned exactly one assembly Lesson a week.
    c. Each Grade must have at least one Lesson scheduled.
    d. A grade cannot have two Lessons scheduled that run on the same day and at the same time.

    e. Each grade is assigned exactly one lunch Lesson a day due to limitations of dining hall size that cannot accommodate all students at the same time. We don't need to substitute for a lunch lesson. We represent "one lesson a day" by making the combination of (grade and day) unique in the relationship set Lunch. Note that, exactly one doesn't imply a key constraint here because day is not a separate entity. If the school runs 5 days a week, a grade will be assigned five lunch lessons for each of those five days. It does imply total participation of Grades. However, we didn't have a way of representing the total participation.
5. A Subject is identified by a name as well as a Grade that it is taught to and the term in which it is taught. If a grade is deleted, we want to delete all Subjects associated with it. Same with deleting a Term. This weak entity set with respect to Term and Grade is represented by the relationship set Must_learn.
    a. Each subject is associated with exactly one Grade and one term.
    b. A subject can be known by many Teachers.
    c. Some subjects (like Writing) require a double lesson once a week. A substitute teacher who is free for both lessons is preferable over two substitute teachers who are only available for one each.
6. A Lesson is identified by the day of the week, as well as the time of the day.
    a. Lunch is available during some lessons only.
    b. Some lessons are taken up by school/grade assembly.
    c. A Lesson can be scheduled in exactly one Room for exactly one Grade with exactly one Teacher.
    d. The timings for lessons must not overlap.
7. A Room is identified by its name. A Room can host many Lessons but not if they run on the same day and same time. A Room may or may not have a Lesson scheduled.
8. Each Term, there is a different schedule. This includes a Teacher might have different half day assigned, Grades might be required to learn a different set of subjects, so subsequently, Teachers might be called on expertise for a different set of subjects.

ER Diagram


Name
Students
ID
Attend
Time
Day
Is_Lunch
Name
Lunch
Grades
Lessons
Is_Assembly
Assembly
Name
Rooms
Schedule
Year
Must_learn
Terms
Half_Day
Teachers
Works_in
Day
Initials
Subjects
Know
Departments
Name
Double_Lessons
Name

ER to Relational Model Translation

```
1. drop table if exists Teachers_know, Schedules, Terms, Lessons,
   Lunch, Rooms, Departments, Subjects, Grades, Students, Terms,
   Teachers;
2. drop type if exists DAYS;
3.
4. create type DAYS as ENUM
5. (
6.       'SUNDAY',
7.       'MONDAY',
8.       'TUESDAY',
9.       'WEDNESDAY',
10.      'THURSDAY',
11.      'FRIDAY',
12.      'SATURDAY'
13.   );
14.
15.   /* entity set Terms */
16.   create table Terms (
17.      year integer primary key,
18.      CHECK ( year > 2021 )
19.   );
20.
21.   /* entity set Lessons */
22.   create table Lessons (
23.      day DAYS,
24.      time tsrange,
25.      primary key (day, time),
26.      is_lunch boolean,
27.      is_assembly boolean,
28.      exclude using gist (time with &&) /* have non-overlapping
   range only for lessons */
29.   );
30.
31.
32.   /* entity set Grades with relationship sets Lunch and Assembly
   */
33.   create table Grades (
34.      name char(32) primary key,
35.      assembly_day DAYS
36.            not null,
37.      assembly_time tsrange,
38.      foreign key (assembly_day, assembly_time) references
   Lessons(day, time),
39.      /* there's only one assembly lesson for a grade */
40.      unique (name, assembly_day)
41.   );
42.
43.
44.   create table Lunch (
```

```
45.      lunch_day DAYS not null,
46.      lunch_time tsrange,
47.      foreign key (lunch_day, lunch_time) references Lessons(day,
   time),
48.      grade char(32) references Grades(name) not null,
49.      primary key(grade, lunch_day, lunch_time)
50.      /* ^ is to ensure each grade gets at most one lunch
51.      lesson per day */
52.   );
53.
54.
55.   /* entity set Students
56.   with relationship set attend */
57.   create table Students (
58.      id varchar(128) primary key, --changed from int to varchar
59.      name char(64),
60.      attend char(32) references Grades(name)
61.            not null    /* total participation */
62.   );
63.
64.   /* entity set Departments */
65.   create table Departments (
66.      name char(32) primary key
67.   );
68.
69.   /* Teachers entity set
70.   with relationship set Works_in and Half_Day */
71.   create table Teachers (
72.      initials char(32) primary key,
73.      half_day DAYS not null,
74.      term int references Terms(year) not null,
75.      department char(32) references Departments(name) not null
76.   );
77.
78.   /* entity set Rooms */
79.   create table Rooms (
80.      room_number integer primary key
81.   );
82.
83.   /* entity set Subjects */
84.   create table Subjects (
85.      name char(32),
86.      double_lesson boolean default false,
87.      term int,
88.      grade char(32),
89.      primary key (name, grade, term),
90.      foreign key (grade) references Grades(name) on delete
   cascade,
91.      foreign key (term) references Terms(year) on delete cascade
92.   );
93.
94.   create table Teachers_know (
95.      teacher char(32),
```

```
96.      subject char(32),
97.      grade char(32),
98.      term int,
99.      foreign key (subject, grade, term) references Subjects(name,
   grade, term),
100.     primary key(subject, grade, term, teacher)
101.  );
102.
103.  /* relationship set Schedule */
104.  create table Schedules (
105.     grade char(32) references Grades(name) not null,
106.     room integer references Rooms(room_number) not null,
107.     teacher char(32) references Teachers(initials) not null,
108.     term int references Terms(year) not null,
109.     day DAYS not null,
110.     time tsrange not null,
111.     foreign key (day, time) references Lessons (day, time),
112.     UNIQUE(day, time, teacher, term), /* one teacher teaches only
   one lesson at a time */
113.     UNIQUE(day, time, room, term), /* one room hosts only one
   lesson at a time */
114.     UNIQUE(day, time, grade, term) /* one grade can attend only
   one lesson at a time */
115.  );
116.
```

Feedback and Revisions

1. Consider adding more attributes other than just the primary key.
    a. Previous version explanation: This is mostly due to the nature of our application and the information it needs. Where needed, non primary key attributes are included.
    b. Current version: We added a student name as an attribute because this would allow us to do some fun queries like “Search for a student by name and show their full profile with their grade, teachers and lesson plan.” While we could have done the same for the teachers, we skipped it because it doesn’t add any new technical challenge. No new attributes have been added to other entity sets because we don’t really need them for the application in mind and we wanted to keep the ER simple where possible.
2. Participation constraint for Teacher-lesson needs to be explained in business rules. Same for grades.
    a. Previous version: We had “A Teacher may or may not have a Lesson scheduled.” in hopes that this explains the relationship. For grades, we had “Each Grade must have a Lesson scheduled.”
    b. Current version: Edited to “A Teacher may or may not have a specific Lesson (day+time) scheduled but will have at least one Lesson scheduled” and for grade, “Each grade must have at least one Lesson scheduled.”
3. If each grade is assigned 1 lunch lesson, this should be represented as total participation. If not, mention in business rules.
    a. Yes. The total participation is indicated by a bold line connecting Grades entity set to Lunch relationship set.
    b. We have edited the Grade entity set description to read:
        i. Each grade is assigned exactly one lunch Lesson a day due to limitations of dining hall size that cannot accommodate all students at the same time. We don’t need a substitute for a lunch lesson. We represent “one lesson a day” by making the combination of (grade and day) unique in relationship Lunch. Note that, “exactly one lesson per day” here doesn’t imply a key constraint because day is not a separate entity. If the school runs five days a week, a grade will be assigned five lunch lessons for each of those five days. It does imply total participation of Grades. However, we didn’t have a way of representing the total participation.
4. Business rules implies that Subjects is a weak entity wrt to Term as well. This is not mentioned in the ER.
    a. As per discussion during the office hours, the relationship set `Must_Learn` represents this relationship. The description has been added to the Subjects entity set as well: “This weak entity set with respect to Term and Grade is represented by the relationship set Must_learn.”

5. Lesson-room-grade is implied to be total participation, but not shown in ER.
    a. Yes, each lesson that a grade has must be held in a room but not every room needs to be hosting a lesson for a grade. This is indicated by a bold line connecting Lesson and Grades to schedule but not room in the ER.
6. Defining foreign keys/not null is not good practice.
    a. We removed `not null` on foreign keys on Subjects because they are also a part of the primary key (with Subjects being a weak entity set). We have some not null requirements on the Schedule entity set as well. While redundant, they are harmless so we are letting them be. The `not null` in the rest of the foreign keys was a way to represent total participation constraints in the relationships.

Data Loading

We received our data as a massive excel sheet from Budhanilkantha School, Nepal with their schedule. The whole schedule is available on request but for now, we haven't attached the full sheet with this submission. For the sake of the project, we ended up taking the schedule of four teachers: AKC, BL, BS, DMS from two different departments over the course of all six days from Sunday through Fridays (schools in Nepal run six days a week). However, we manufactured the students that these teachers would potentially teach– partly because we do not have the student data and partly because we do not want the student data for privacy reasons. After the initial acquisition of the data, we had to refine them through multiple steps/stages before we could use them:
Step 1: Select a subset of the data for the proof of concept.
Step 2: Reorganize this subset of data from a excel sheet which was difficult to handle via custom python scripts into more structured format then save as data.csv
Step 3: Write a python script to read the schema.sql file to create tables then to read the data.csv file to create insert statements that load data into the tables.

We provide a load.sql file that any user can run with `psql -U pa1038 -d pa1038_db -f load.sql` and get the data loaded. We also provide a python script called create-data.py that can be used to generate this load.sql file with `python3 create-data.py > load.sql`

All the relevant files including schema.sql with create table statements, data.csv file with the used subset of the data and create-data.py file with necessary python code to generate insert statements are available in the project github repo
https://github.com/prasantadh/principles-of-db-f22

Application Details and User Interactions

Our webpage is inclusive of six main tabs:

1. Teachers
   This page provides the user with all information about individual teachers. The user is prompted to select a teacher from a dropdown menu and upon doing so, information about the respective teachers' class timings and schedule, number of students and number of classes is relayed back to the user.

2. Students
   This tab provides the user with all relevant information about individual students. After selecting a student from a dropdown list, the students' schedule for the week, the subjects they have, the number of classes they have in a week and their lunch and assembly timings are displayed.

3. Lessons
   The Lessons page provides a schedule of class timings on particular days. The user will be able to select a day of the week after which they are shown all the grades, teachers, classrooms and class timings for that day.

4. Grades
   The Grades tab displays the number of students in each grade and also a weekly schedule with the name of the teachers that are teaching that grade and the lesson timings for each individual grade.

5. Rooms
   The tab Rooms allows a user to search a room and check its occupancy/availability. It shows the classroom number along with which grade and teacher is using it and the duration of their lesson. There is also a view that shows the number of classes that take place in a specific room throughout the week.

6. Find Substitute
   The highlight of our project, the Find Substitute page allows a user to input an absent teacher, and find all other possible substitute teachers for specific lessons and lesson timings. The user can select multiple teachers that are absent today (by default today's day is taken) or teachers that will be absent on a particular day, and find other available teachers that can take their class timings. Since we took a smaller subset of the overall data we had, we only have four teachers making substitution for two teachers a bit of a challenge but is an easy fix by including more tuples and data.

The comments in our code highlight where we have made use of JOIN and GROUP BY statements in our queries. We have 5 main JOIN statements and 4 GROUP BY queries.