University Dorm Management System

Port number: 8606

http://128.238.64.108:8606/

By

Rigveda Vangipurapu - rv2205 | Sindhu Bhoopalam Dinesh - sb8019

Description:

Universities have a large number of student dorm-related data every year. Each student has to apply for room occupancy, make rent payments, register complaints in case of any issues, and all the data has to be tracked by employees/admins to manage and maintain the dorms. So an efficient dorm management database system can be a one-stop application for everyone involved in the maintenance of the dorms that makes the data storage and management easy and less cumbersome.

Our application consists of 3 main users:

I. Students
II. Employees
III. Admin

The user interface for the application is as follows (questions the user can ask/views available):

I. Students Module

The students can use the application to perform the following tasks:

1. Registration - Students can add in their basic details to fill up the profile.
2. Application - Once registered, students can fill in a waitlist application with their preferences to request occupancy in the dorm and also check the status of the application.
3. Request/Complaint - Students who are occupying the dorm rooms can make a request/complaint pertaining to any issues in the room based on categories - hallway, kitchen, electricity, pest control, and so on.
   Further, the student can also update the status of this complaint, once resolved.
   Ex: A broken light in the kitchen
4. Payment - Students can use this task to pay their rent and also fetch reports of their payments.
5. Dining - Students can check the open cafes, their menus, timings and also use the order-in feature to get food delivered to their rooms.

II. Employees Module

Employees could be anyone involved in the maintenance of the dorms - janitor, housekeeping, electrician, etc. Employees can use the application for the following tasks:

1. Registration - Employees can add in their basic details to fill up the profile.
2. Tasks - Employees can view the tasks assigned to them. He can also view the previously completed tasks.
3. Payment - Employees can check their paycheck status.

III. Admins Module

Admin can use the application for the following tasks:

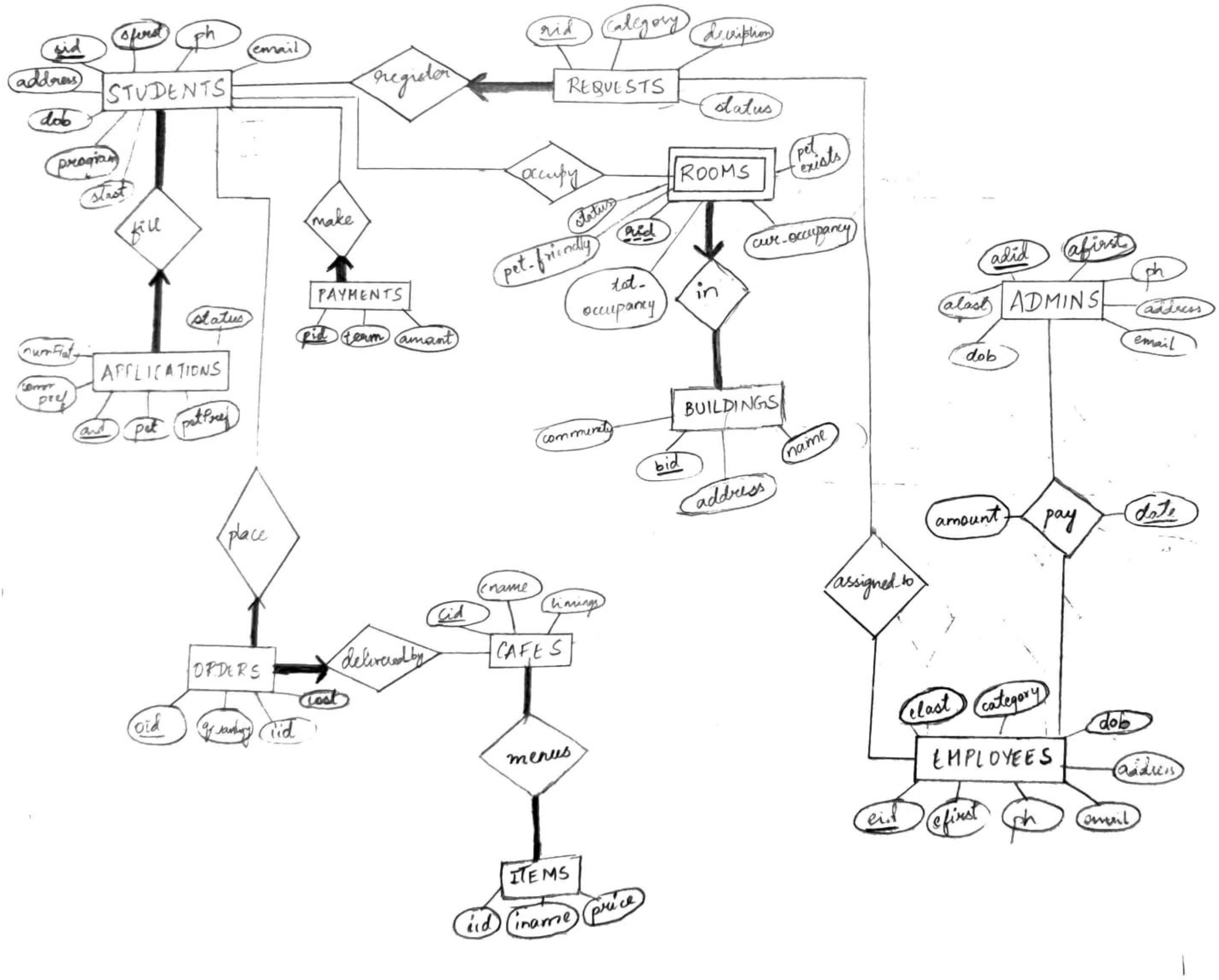
1. Registration - Admins can add in their basic details to fill up the profile.
2. Manage Dorms - The admin can view dorms/rooms, update the dorm occupancy, vacancies, etc. He/She can further assign dorms to students based on their requests.
3. Assign tasks - After viewing the requests/complaints of the students, the admin can assign the task to an employee based on the category.
4. Manage student payment activity - The admin can check the payment status of student rents, deposits and generate reports.
5. Manage employee payroll - The admin can register and update payments done to the employees.

All errors on the application are handled and even shown for further debugging.

Business Rules for this application are:

1) Each student must fill in an application form to apply for room occupancy exactly once.
2) Each application is filled by only one student.
3) Each payment done relates to a single student.
4) Each request/complaint made relates to a single student.
5) A student may register a request/complaint and update the status to 'completed' once the issue is resolved. Else the status remains 'in progress'.
6) Each order placed relates to a single student.
7) Each order is taken up for preparation by just one cafe. A cafe can make any number of orders.
8) Each room is located in a single building. A building has at least one room. If a building is destroyed, the rooms in it will be destroyed too.
9) A room or a building cannot be deleted as long as it is not totally unoccupied.
10) Each employee may or may not be assigned student requests.
11) Each request can be assigned to one employee, if assigned.
12) Admin pays salaries to the employees who have completed the task. Each admin may or may not manage salary payment.
13) While placing orders from the cafe, in order to buy multiple items, students have to place multiple orders.
14) After each term, we assume that the database is restructured by the Database Administrator in bulk to account for the students who left and any changes in rooms.

The ER diagram for the application:



The application has the following entity and relationship sets:

1. Students: Contains the following information about the student-student ID, student first name, student last name, phone number, email ID, address.
2. Applications: Students fill out applications in order to get rooms allocated. It contains the following- application ID, number of roommates preferred, student ID corresponding to the student who filled the application, pet details if any, pet preferences, preferred community of housing.

3. Payments: Students pay their bills and dues which are recorded in this table. It contains the following information- payment ID, term, amount, student ID corresponding to the student who did the payment.
4. Cafes: The University has multiple cafes, and the following details are stored in this table- cafe ID, Cafe name, working hours.
5. Orders: Students can place orders which are fulfilled by the cafes. The Orders table stores the following- order ID, quantity, item ID, cost, the ID of the student who placed the order, the ID of the cafe fulfilling the order.
6. Items: Holds the items available at various cafes with item ID, item name, and price.
7. Menus: Holds item ID and corresponding cafe ID in which the item is present.
8. Requests: The students fill in various requests related to dorms and dining which are stored at this table. It has the following details- request ID, category, description, the ID of the student filling in the request.
9. Buildings: Contains the following information about buildings in the university- building ID, building name, address, community in which the building exists.
10. Rooms_In: Contains the information about rooms- room ID, total occupancy capacity, current occupancy, the status of the room, the ID of the building in which the room is located, pet-friendly information, pet existence information,
11. Employees: Contains the following information about the employee- employee ID, employee first name, employee last name, phone number, email ID, address.
12. Pay: Contains the following information about employee salaries- amount, the date on which the payment is initiated, the ID of the employee corresponding to the payment made and the ID of the admin making the payment.
13. Assigned_to: Student requests are assigned to employees, so the table stores the employee ID and the corresponding request ID that he/she has been assigned.
14. Admins: Contains the following information about the admin- admin ID, admin name, phone number, email ID, and address.

Data loading procedure

The data for this application was self-constructed and keeps adding up on the go. We started off with a CSV file of around 20 entries per table.

The following commands were used to load the table from the CSV files.

```
cat Students.csv | psql -U rv2205 -d rv2205_db -c "COPY Students from STDIN CSV HEADER"
cat Requests.csv | psql -U rv2205 -d rv2205_db -c "COPY Requests from STDIN CSV HEADER"
cat Admins.csv | psql -U rv2205 -d rv2205_db -c "COPY Admins from STDIN CSV HEADER"
cat Applications.csv | psql -U rv2205 -d rv2205_db -c "COPY Applications from STDIN CSV HEADER"
cat Employees.csv | psql -U rv2205 -d rv2205_db -c "COPY Employees from STDIN CSV HEADER"
cat Assigned_To.csv | psql -U rv2205 -d rv2205_db -c "COPY Assigned_To from STDIN CSV HEADER"
cat Items.csv | psql -U rv2205 -d rv2205_db -c "COPY Items from STDIN CSV HEADER"
cat Menus.csv | psql -U rv2205 -d rv2205_db -c "COPY Menus from STDIN CSV HEADER"
cat Pay.csv | psql -U rv2205 -d rv2205_db -c "COPY Pay from STDIN CSV HEADER"
cat Payments.csv | psql -U rv2205 -d rv2205_db -c "COPY Payments from STDIN CSV HEADER"
cat Rooms_in.csv | psql -U rv2205 -d rv2205_db -c "COPY Rooms_in from STDIN CSV HEADER"
cat Buildings.csv | psql -U rv2205 -d rv2205_db -c "COPY Buildings from STDIN CSV HEADER"
cat Cafes.csv | psql -U rv2205 -d rv2205_db -c "COPY Cafes from STDIN CSV HEADER"
```

Schema

```
drop table if exists Applications cascade;
drop table if exists Payments cascade;
drop table if exists Orders cascade;
drop table if exists Cafes cascade;
drop table if exists Requests cascade;
drop table if exists Rooms_In cascade;
drop table if exists Buildings cascade;
drop table if exists Occupy cascade;
drop table if exists Students cascade;
drop table if exists Pay cascade;
drop table if exists Admins cascade;
drop table if exists Assigned_To cascade;
drop table if exists Employees cascade;
drop table if exists Items cascade;
drop table if exists Menus cascade;

create table Students (
sid integer primary key,
sfirst varchar(128) not null,
slast varchar(128) not null,
ph char(10),
email varchar(64),
address varchar(512),
dob varchar(16),
dept varchar(32),
program varchar(32)
);

create table Applications(
aid serial primary key,
sid integer not null,
pet varchar(3),
petPref varchar(3),
numFlat integer,
commPref char(1),
status varchar(16),
foreign key( sid ) references Students( sid )
);

create table Payments(
pid serial primary key,
term varchar(4),
amount decimal,
sid integer not null,
foreign key( sid ) references Students( sid )
);

create table Cafes(
cid integer primary key,
name varchar(128) not null,
timings varchar(32)
);

create table Items(
iid serial primary key,
iname varchar(32),
price decimal
);

create table Menus(
iid integer,
cid integer,
primary key( iid, cid ),
foreign key( iid ) references Items( iid ),
foreign key( cid ) references Cafes( cid )
);

create table Orders(
oid serial primary key,
quantity integer not null,
iid integer not null,
cost decimal,
sid integer not null,
cid integer not null,
foreign key( sid ) references Students( sid ),
foreign key( cid ) references Cafes( cid )
);

create table Requests(
rid serial primary key,
category varchar(32),
description varchar(128),
sid integer not null,
status varchar(16),
foreign key( sid ) references Students( sid )
);
```

```
create table Buildings (
bid integer primary key,
name varchar(64),
community char(1),
address varchar(128) not null
);

create table Rooms_In(
rid integer primary key,
tot_occupancy integer,
cur_occupancy integer,
pet_friendly varchar(3),
pet_exists varchar(3),
bid integer not null,
foreign key (bid) references Buildings( bid ) on delete cascade
);

create table Occupy(
sid integer,
rid integer,
bid integer,
primary key(sid, rid, bid ),
foreign key (sid) references Students( sid ),
foreign key (rid) references Rooms_In( rid ),
foreign key (bid) references Buildings( bid )
);

create table Employees(
eid integer primary key,
efirst varchar(128) not null,
elast varchar(128) not null,
dob varchar(16),
ph char(10),
email varchar(64),
address varchar(512),
category varchar(32)
);

create table Assigned_To(
eid integer,
rid integer,
primary key(eid,rid),
foreign key (eid) references Employees(eid),
foreign key (rid) references Requests(rid)
);
```

```
create table Admins (
adid integer primary key,
afirst varchar(128) not null,
alast varchar(128) not null,
dob varchar(16),
ph char(10),
address varchar(512),
email varchar(64)
);

create table Pay(
adid integer,
eid integer,
date varchar(16),
amount decimal,
primary key(adid,eid, date),
foreign key (adid) references Admins(adid),
foreign key (eid) references Employees(eid)
);

alter sequence applications_aid_seq restart with 500;
alter sequence requests_rid_seq restart with 500;
alter sequence payments_pid_seq restart with 500;
```

Note: Starting the sequence with a custom serial to avoid clashes with already inserted data

The following constraints could not be captured:

- The participation constraint of Students in the relationship fills.
- The participation constraint of Buildings in the relationship in.
- The participation constraint of Cafes in the relationship menus.
- The participation constraint of Items in the relationship menus.