Name of Application:
Bathroom Tracker

Team Members:

Matthew Bentley
David Lance
Matt Roseman

Application Background:

The purpose of this application is to give a rating system for bathrooms on CWRU's campus. The primary use case would be for students to find the best bathrooms on campus and to advise others as to the current conditions of given bathrooms.

Interfaces:

- Login functionality through Case
- Viewing individual bathrooms and their reviews
- Add review page
- Add bathroom page
- Add building with times open
- Add major
- Add major building entrance
- View bathrooms in a building
- Set user's major
- All buildings
- All buildings accessible currently
- All buildings available to user's major
- View bathrooms with above x stars
- View bathrooms with above x stars in the last week
- View bathrooms with above x stars in the last month

Data Description:

Buildings

Data
-id of building (int)
-name of building (string)
-building open time (time) - refers to the time at which the building is open to all students meaning that a person with a major as defined in the building_access table will have access 24/7
-building close time (time) - refers to the time the building is closed to nonmajors

Constraints

-building open time must come before building close time

Majors

Data

-id of major (int)

-name of major (string)

Constraints

Bathrooms

Data

-id of bathroom (int)

-gender of bathroom (gender enum)

-type of bathroom (type enum)

Constraints

Users

Data

-case id of user (string)

-last time the user was seen on the site (datetime)

Constraints

-last seen must be a datetime in the past

Reviews

Data

-the review comment (string)

-the review rating (int)

-when the review was left (datetime)

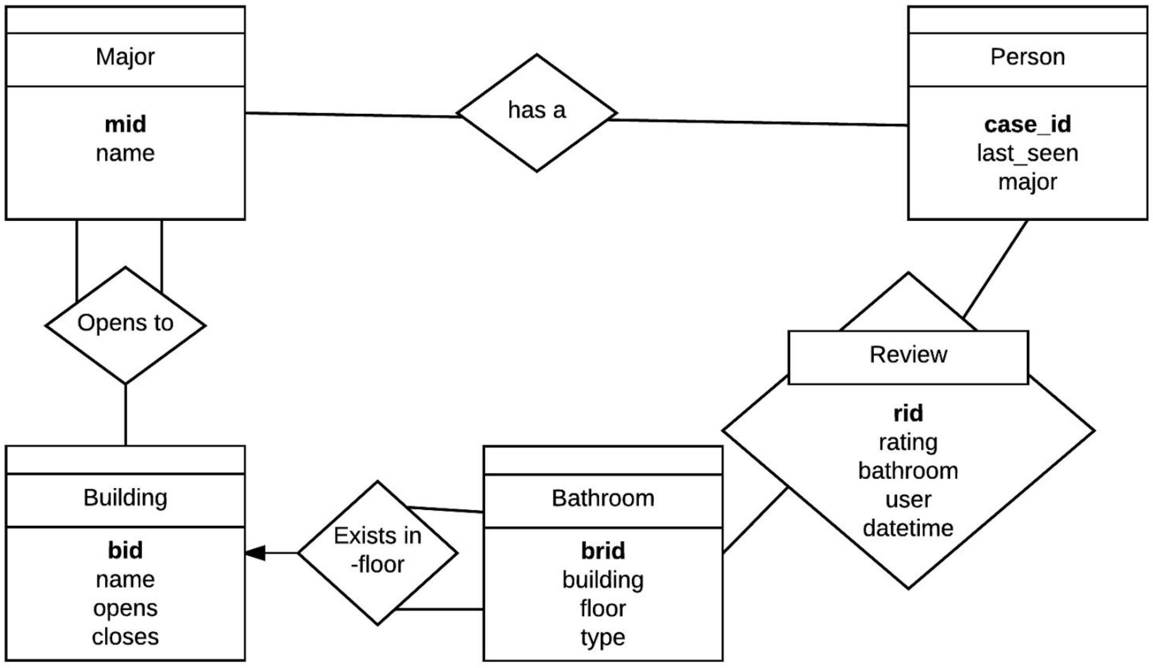
Constraints

-review comments must be under 10000 characters

-the review time must be a past datetime

-rating can only be numbers between 1 and 5

ER diagram:



Bold = Primary/Candidate Key

<u>Function Dependencies:</u>

Major(<u>mid</u>, name)

- mid →name
- mid is the only key therefore BCNF

Building(<u>bid</u>, name, opens, closes)

- bid→name, opens, closes
- bid is the only key therefore BCNF

Building Access(<u>major</u>, <u>building</u>)

- All attributes are keys therefore BCNF

Person: (<u>case_id</u>, last_seen, major)

- case_id→ last_seen, major
- case_id is the only key therefore BCNF

Bathroom: (<u>brid</u>, building,floor, type)

- brid→ building, floor, type
- brid is the only key therefore BCNF

Review: (<u>rid</u>, rating, review, bathroom, user, datetime)

- rid→ rating, review, bathroom, user, datetime
- rid is the only key therefore BCNF

<u>Schema:</u>

Major(mid: int, name: string)
Building(bid: int, name: string, opens: datetime, closes: datetime,)
Building_Access(mid: int, bid: int)
Person(case_id: string, last_seen: datetime, mid: int)
Bathroom(brid: int, bid: int, floor: int, type: gender)
Review(rid: int, rating: int, review: string, bid: int, case_id: string, datetime: datetime)

Implementation:

Overview:
Our application tracks bathrooms across campus. Users can add bathrooms, buildings, majors, and users, and the application can help users find open bathrooms given their major and the current time of day, as well as track reviews and ratings for various bathrooms.

Application Requirements:
The application is rather simple overall: users log in using the CWRU CAS login. They can then add buildings, majors, and bathrooms. Additionally, they can assign themselves a major, and associate majors with buildings. Buildings are considered open between the 'opens' and 'closes' time stored with the building. The application can also show users what buildings they have 24 hour access to, given their major and the majors associated with buildings. Finally, users can provide ratings for bathrooms (between 1 and 5) as well as text reviews. The average rating is shown when bathrooms are listed, and reviews are shown on specific bathroom pages.

Database Requirements:
Specifically, the database stores a table of buildings which includes when it opens and closes, a table of majors, a table associating buildings to majors in a many to many relationship, a table of users that keeps track of their major, a table of bathrooms that keeps track of their building, floor, and gender, and a table of bathroom reviews with the review, rating, user, and obviously bathroom. Simple queries include adding buildings, majors, users, etc; as well as listing reviews associated with a bathroom or bathrooms associated with a building. More complicated queries include getting a list of all bathrooms a user currently has access to and listing bathrooms with the average rating.

Creating Database:
This application requires PostgreSQL version 9.5 or greater. We make use of Postgres specific functionality, specifically the UPSERT feature to update or insert in one command. We did not use a GUI, just the psql command line.

```
CREATE DATABASE bathroom_tracker;
CREATE USER bathroom PASSWORD 'bath';
GRANT ALL ON DATABASE bathroom_tracker TO bathroom;
CREATE TABLE major (
      mid INTEGER PRIMARY KEY,
     name VARCHAR(64) NOT NULL
);
CREATE TABLE building (
     bid INTEGER PRIMARY KEY,
     name VARCHAR(64) NOT NULL,
     opens TIME WITHOUT TIME ZONE,
     closes TIME WITHOUT TIME ZONE
);
CREATE TABLE building_access (
     building INTEGER REFERENCES building(id),
     major INTEGER REFERENCES major(id),
     PRIMARY KEY (building, major)
);
CREATE TABLE person (
     case_id VARCHAR(8) PRIMARY KEY,
     major INTEGER REFERENCES major(id),
     last_seen TIMESTAMP WITHOUT TIME ZONE
);
CREATE TYPE gender AS ENUM (‘male’, ‘female’, ‘neither’);
CREATE TABLE bathroom (
     brid INTEGER PRIMARY KEY,
     building INTEGER REFERENCES building(id) NOT NULL,
     floor INTEGER,
     gender gender
);
CREATE TABLE review (
     rid INTEGER PRIMARY KEY,
     bathroom INTEGER REFERENCES bathroom(id),
     review TEXT,
     rating INTEGER CHECK (rating > 0 AND rating <= 5)
     time_added TIMESTAMP WITHOUT TIME ZONE
);
```

<u>Queries:</u>

Query 1:

- List all the bathrooms along with the building name that they are in, the floor of that building they reside on, and the gender of the bathroom.

SQL:

```
     SELECT brid, floor, gender, name
     FROM bathroom NATURAL JOIN building;
```

RA:

$\boldsymbol{\Pi}_{\text{brid, floor, gender, name}}(\text{building} \bowtie \text{bathroom})$

TRC:

{$t^{(4)}$ | ∃(br)Bathroom(br) ^ ∃(b)Building(b) ^ br.bid = b.bid ^ t(brid) = br.brid ^ t(floor) = br.floor ^ t(gender) = br.gender ^ t(name) = b.name}

Query 2:

- List all the bathrooms that are currently accessible to all students.
- Given: current timestamp

SQL:

```
SELECT brid, floor, gender
FROM bathroom NATURAL JOIN building
WHERE *current timestamp* > opens AND *current timestamp* < closes;
```

RA:

$\boldsymbol{\Pi}_{\text{brid, floor, gender}}(\sigma_{\text{opens < *current\_timestamp* ^ *current timestamp* < closes}}(\text{building} \bowtie \text{bathroom}))$

TRC:

{$t^{(3)}$ | ∃(br)Bathroom(br) ^ ∃(b)Building(b) ^ br.bid = b.bid ^ t(brid) = br.brid ^ t(floor) = br.floor ^ t(gender) = br.gender ^ *current timestamp* > b.opens ^ *current timestamp* < b.closes}

Query 3:

- List all the bathrooms that are currently accessible to the current user.
- Given: current timestamp, case_id

SQL:

```
SELECT * FROM bathroom, building, building_access, person
WHERE person.case_id = *case_id* AND bathroom.building=building.id AND
building_access.building=building.id AND
(person.major = building_access.major OR (*current_time* > building.opens AND
 *current_time* <  building.closes));
```

RA:

$\boldsymbol{\Pi}_{\text{brid, floor, gender}}((\sigma_{\text{opens < *current\_timestamp* ^ *current timestamp* < closes}}(\text{building} \bowtie \text{bathroom})) \cup$
$(\sigma_{\text{case\_id = *case\_id*}}(((\text{building} \bowtie \text{bathroom}) \bowtie \text{building\_access}) \bowtie \text{person})))$

TRC:

{$t^{(3)}$ | ∃(br)Bathroom(br) ^ ∃(b)Building(b) ^ br.bid = b.bid ^ t(brid) = br.brid ^ t(floor) = br.floor ^ t(gender) = br.gender ^
((*current timestamp* > b.opens ^ *current timestamp* < b.closes) ∨
(∃(p)Person(p) ^ ∃(ba)Building_Access(ba) ^ ba.bid = b.bid ^ p.mid = ba.mid ^ p.case_id = *case_id*))}

Query 4:

- List all bathrooms in a given building.

- GIven: bid

SQL:

```
SELECT brid, floor, gender
FROM bathroom NATURAL JOIN building
WHERE bid=*bid*;
```

RA:

$\Pi_{\text{brid, floor, gender}}(\sigma_{\text{bid = *bid*}}(\text{building} \bowtie \text{bathroom})$

TRC:

{$t^{(e)}$ | ∃(br)Bathroom(br) ^ ∃(b)Building(b) ^ br.bid = b.bid ^ t(brid) = br.brid ^ t(floor) = br.floor ^ t(gender) = br.gender ^ *bid* = b.bid}

Query 5:

- List all the reviews of bathrooms(For SQL find all the bathrooms with an average rating above a given rating)

SQL:

```
SELECT brid, floor, name, gender, AVG(rating)
FROM (bathroom NATURAL JOIN building) NATURAL JOIN review
GROUP BY brid;
```

RA:

$\Pi_{\text{brid, floor, gender, name, rating}}((\text{building} \bowtie \text{bathroom}) \bowtie \text{review})$

TRC:

{$t^{(5)}$ | ∃(br)Bathroom(br) ^ ∃(b)Building(b) ^ ∃(r)Review(r) ^ br.bid = b.bid ^ t(brid) = br.brid ^ t(floor) = br.floor ^ t(gender) = br.gender ^ t(name) = b.name ^ r.brid = br.brid ^ t(rating) = r.rid}

Query 6:

- List all the reviews of a bathroom
- Given: brid

SQL:

```
SELECT rating, review FROM review
WHERE bathroom=*brid*
```

RA:

$\Pi_{\text{rating, review}}(\sigma_{\text{brid=*brid*}}(\text{bathroom} \bowtie \text{review}))$

TRC:

{$t^{(2)}$ | ∃(br)Bathroom(br) ^ ∃(r)Review(r) ^ r.brid = br.brid ^ br.brid = *brid* ^ t(rating) = r.rid ^ t(review) = r.review}

Query 7:

- List all the reviews with ratings in the last week with their building names, gender and floor. (For SQL find the averages of the bathrooms in that week that are above a given rating)

- Given: 1 week ago timestamp

SQL:

```
SELECT brid, floor, name, AVG(rating)
FROM (bathroom NATURAL JOIN building) NATURAL JOIN review
GROUP BY brid
HAVING time_added > *1 week ago timestamp*;
```

RA:

$\Pi_{\text{brid, floor, gender, name, rating}}(\sigma_{\text{time\_added > *1 week ago timestamp*}}(\text{building} \bowtie \text{bathroom}) \bowtie \text{review})$

TRC:

$\{t^{(5)} \mid \exists(br)\text{Bathroom}(br) \wedge \exists(b)\text{Building}(b) \wedge \exists(r)\text{Review}(r) \wedge br.bid = b.bid \wedge r.brid = br.brid \wedge t(brid) = br.brid \wedge t(floor) = br.floor \wedge t(gender) = br.gender \wedge t(name) = b.name \wedge t(rating) = r.rid \wedge r.time\_added > \text{*1 week ago timestamp*}\}$

Roles and Contributions:

Matthew Bentley
Wrote a large part of the code and a few smaller parts of this document, including the first draft of the overview and the manuals. Deployed the project onto people.acm.case.edu:8080
David Lance
Responsible for a majority of the query documentation and assisted in the development of the database schema and its constraints.
Matthew Roseman
Helped in extending the views available through the application and helped in refining the schema documentation.

What we learned:
This project showcased to us the advantages of actively defining a database schema. While we were developing new ideas arose when looked at what specific joins would display rather than looking for outputs we were given them on a silver platter. By and large this taught that although something that abstracts the queries away is convenient and quick they do not always open your eyes to the possibilities your data can bring.

Installation Manual
Installation requires python 3, virtualenv, and Postgres libraries (libpg-dev on Debian).
First, create a virtualenv with `pyvenv venv`
Then enter the virtualenv with `source venv/bin/activate`
Next, install the requirements with `pip install -r require ments.txt`
Export the database connection info into the environment `export PGCONN=”dbname=bathroom_tracker user=bathroom password=bath host=localhost”`
Finally, after creating the tables in the database (from create.sql), run the server with `python server.py`

User's Manual:

Use of the program is fairly straightforward: once it’s running, navigate to its web address and follow the links on the main page to add buildings, bathrooms, majors, and more.
The application works best in Google Chrome.

Programmer’s Manual:

The application itself is also very simple. Just a number of functions that run queries and return text. A few use templates, which makes returning HTML easier.