

CS 452 AI SQL Write-up

Spooky Mansion AI SQL Project

GitHub

You can find the source code for this project [here](#)

Schema Summary

Essentially, I just made an SQL schema that models a game with players that explore a spooky mansion and can collect items and escape to get to the next floor. They also encounter monsters with abilities and ghosts that can attack them, and they work together to get to the next floor of rooms.

Below is my actual sql setup file that was used to setup the tables:

```
create table public.player (
  player_id bigint generated by default as identity not null,
  created_at timestamp with time zone not null default now(),
  username character varying not null default 'user'::character varying,
  health real not null default '10'::real,
  sanity real not null default '10'::real,
  birthday date null,
  constraint player_pkey primary key (player_id),
  constraint player_username_key unique (username)
) TABLESPACE pg_default;

create table public.item (
  item_id bigint generated by default as identity not null,
  created_at timestamp with time zone not null default now(),
  name character varying not null default 'itemx'::character varying,
  description character varying null,
  constraint item_pkey primary key (item_id),
  constraint item_item_id_fkey foreign KEY (item_id) references player
(player_id)
) TABLESPACE pg_default;

create table public.item_owner (
  owner_id bigint not null,
  created_at timestamp with time zone not null default now(),
  item_id bigint null,
  id bigint generated by default as identity not null,
  constraint item_owner_pkey primary key (id),
  constraint item_owner_item_id_fkey foreign KEY (item_id) references item
```

```
(item_id),
    constraint item_owner_owner_id_fkey foreign KEY (owner_id) references
player (player_id)
) TABLESPACE pg_default;
```

```
create table public.room (
    room_id bigint generated by default as identity not null,
    created_at timestamp with time zone not null default now(),
    name character varying not null,
    description character varying not null,
    lights_on boolean not null default true,
    constraint room_pkey primary key (room_id)
) TABLESPACE pg_default;
```

```
create table public.ghost (
    ghost_id bigint generated by default as identity not null,
    created_at timestamp with time zone not null default now(),
    name character varying null,
    favorite_food character varying null,
    sanity_attack_power real not null default '1'::real,
    constraint ghost_pkey primary key (ghost_id)
) TABLESPACE pg_default;
```

```
create table public.monster (
    monster_id bigint generated by default as identity not null,
    created_at timestamp with time zone not null default now(),
    name character varying not null default 'zombie'::character varying,
    health real null,
    attack_power real not null default '5'::real,
    defense_power real null,
    constraint monster_pkey primary key (monster_id)
) TABLESPACE pg_default;
```

```
create table public.ability (
    ability_id bigint generated by default as identity not null,
    created_at timestamp with time zone not null default now(),
    name character varying not null,
    description character varying not null,
    danger_level integer not null,
    constraint ability_pkey primary key (ability_id)
) TABLESPACE pg_default;
```

```
create table public.monster_ability (
    monster_id bigint not null,
    created_at timestamp with time zone not null default now(),
    ability_id bigint null,
    id bigint generated by default as identity not null,
    constraint monster_ability_pkey primary key (id),
    constraint monster_ability_ability_id_fkey foreign KEY (ability_id)
references ability (ability_id),
```

```

    constraint monster_ability_monster_id_fkey foreign KEY (monster_id)
references monster (monster_id)
) TABLESPACE pg_default;

create table public.room_belongings (
    room_id bigint not null,
    created_at timestamp with time zone not null default now(),
    player_id bigint null,
    item_id bigint null,
    monster_id bigint null,
    ghost_id bigint null,
    id bigint generated by default as identity not null,
    constraint room_belongings_pkey primary key (id),
    constraint room_belongings_ghost_id_fkey foreign KEY (ghost_id)
references ghost (ghost_id),
    constraint room_belongings_item_id_fkey foreign KEY (item_id) references
item (item_id),
    constraint room_belongings_monster_id_fkey foreign KEY (monster_id)
references monster (monster_id),
    constraint room_belongings_player_id_fkey foreign KEY (player_id)
references player (player_id),
    constraint room_belongings_room_id_fkey foreign KEY (room_id) references
room (room_id)
) TABLESPACE pg_default;

```

Below is the sql file I used to insert dummy data into the tables:

```

INSERT INTO public.player (player_id, username, health, sanity, birthday)
VALUES
(1, 'GhostHunter99', 100, 90, '1990-05-15'),
(2, 'KnightOfLight', 80, 70, '1985-11-10'),
(3, 'ShadowSlayer', 75, 60, '1997-03-20'),
(4, 'FireMage21', 90, 65, '1992-09-05'),
(5, 'IceQueen88', 85, 85, '2001-01-30'),
(6, 'SpectralRogue', 95, 75, '1998-08-12');

INSERT INTO public.item (item_id, name, description) VALUES
(1, 'Sword of Flames', 'A sword imbued with the power of fire'),
(2, 'Shield of Eternity', 'A shield that protects against all harm'),
(3, 'Boots of Speed', 'Grants the wearer unparalleled agility'),
(4, 'Amulet of Wisdom', 'An artifact that enhances intelligence'),
(5, 'Potion of Energy', 'Restores stamina and boosts strength
temporarily');

INSERT INTO public.item_owner (id, owner_id, item_id, created_at) VALUES
(1, 1, 1, now()),
(2, 2, 2, now());

INSERT INTO public.room (room_id, name, description, lights_on) VALUES

```

```
(1, 'Grand Hall', 'A massive hall with high ceilings and chandeliers',
true),
(2, 'Dungeon', 'A dark and damp underground area', false),
(3, 'Library', 'A quiet place filled with ancient books', false),
(4, 'Conservatory', 'A glass room filled with exotic plants', true),
(5, 'Servants' Quarters', 'A small and modest living area for servants',
false);
```

```
INSERT INTO public.ghost (ghost_id, name, favorite_food,
sanity_attack_power) VALUES
(1, 'Spooky Sally', 'Haunted Honeycakes', 5),
(2, 'Phantom Paul', 'Ectoplasmic Pudding', 7),
(3, 'Wailing Wanda', 'Ghostly Grapes', 6);
```

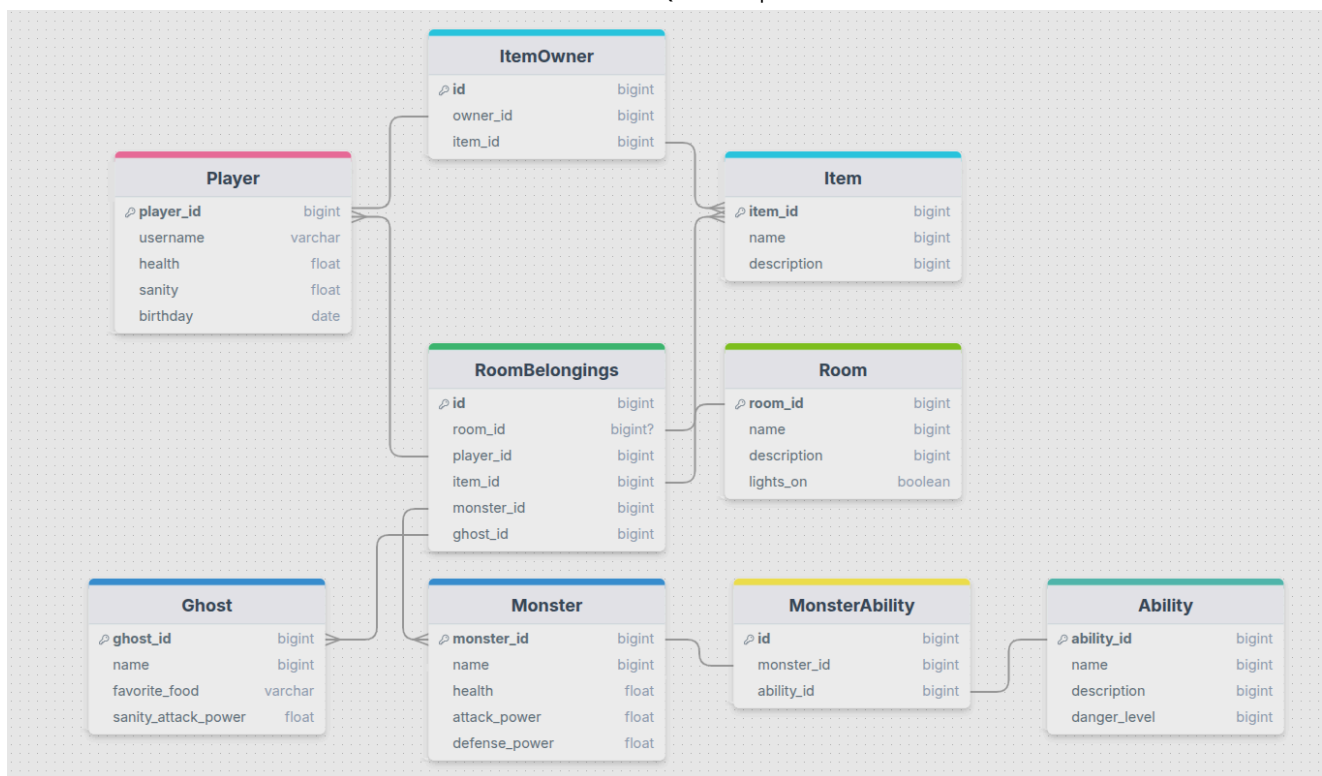
```
INSERT INTO public.monster (monster_id, name, health, attack_power,
defense_power) VALUES
(1, 'Shadow Beast', 120, 15, 5),
(2, 'Fire Golem', 200, 20, 10),
(3, 'Ice Serpent', 150, 25, 8);
```

```
INSERT INTO public.ability (ability_id, name, description, danger_level)
VALUES
(1, 'Fire Breath', 'A powerful attack that burns everything in its path',
5),
(2, 'Ice Shard', 'Launches sharp, freezing projectiles', 4),
(3, 'Shadow Cloak', 'Grants temporary invisibility', 3);
```

```
INSERT INTO public.monster_ability (id, monster_id, ability_id,
created_at) VALUES
(1, 1, 3, now()),
(2, 2, 1, now()),
(3, 3, 2, now());
```

```
INSERT INTO public.room_belongings (id, room_id, player_id, item_id,
monster_id, ghost_id, created_at) VALUES
(1, 1, 1, 1, NULL, NULL, now()),
(2, 2, 3, 5, 1, NULL, now()),
(3, 3, 2, NULL, NULL, 1, now()),
(4, 4, 4, NULL, NULL, NULL, now()),
(5, 5, 6, 3, NULL, NULL, now()),
(6, 2, NULL, 4, 2, NULL, now()),
(7, 1, 5, NULL, NULL, NULL, now());
```

And finally, here is the picture of the database design I drew on [Draw SQL](#):



Queries I Tried

I tried a nice handful of questions for querying my database involving things like which players have items, where a certain item not picked up by a player is in the mansion, and which room has ghosts and their names and favorite foods.

Prompting Strategies I Tried

I tried just 2 prompting strategies (*zero-shot prompting* and *single-domain-double-shot prompting*). The *single-domain-double-shot prompting* did do slightly better, not in terms of accuracy, but just slightly more detailed responses, but overall the 2 prompting methods did about the same. I think there may have been a bigger gap between the two if I would have had slightly more complex queries to run, or I could have tried a *multi-shot* approach to give a wide variety of example questions, sql output, etc.

Query That Went Well

Question

"Which rooms have ghosts in them and what are the ghost's names and favorite foods?"

GPT SQL Output

```

SELECT r.room_id, r.name, g.name AS ghost_name, g.favorite_food
FROM room r
JOIN room_belongings rb ON r.room_id = rb.room_id
JOIN ghost g ON rb.ghost_id = g.ghost_id;
  
```

queryRawResponse

"[(3, 'Library', 'Spooky Sally', 'Haunted Honeycakes')]"

Friendly NLQ Response

"The Library has a ghost named Spooky Sally who loves Haunted Honeycakes."

Queries That Didn't Go Too Well

I actually have 2 that didn't go well. One that constantly errored (although if I provided a good example prompt for this I don't believe this would have been the case), and another one that didn't correctly do the right query.

The second question listed below says no room has players in it, but in reality the only room that doesn't have players in it is the Dungeon.

Question 1

"Which rooms still have their lights turned on?"

GPT SQL Output

```
SELECT r.room_id, r.name
FROM room r
WHERE r.lights_on = 1;
```

queryRawResponse

"[(3, 'Library', 'Spooky Sally', 'Haunted Honeycakes')]"

Friendly NLQ Response

"N/A Due To Error"

Error

"operator does not exist: boolean = integer\nLINE 4: WHERE lights_on = 1;\n ^\nHINT: No operator matches the given name and argument types. You might need to add explicit type casts.\n"

Question 2

"Which room has no players in it?"

GPT SQL Output

```
SELECT r.room_id, r.name
FROM room r
LEFT JOIN room_belongings rb ON r.room_id = rb.room_id AND rb.player_id IS
NOT NULL
WHERE rb.player_id IS NULL;
```

queryRawResponse

"[]"

Friendly NLQ Response

"No room has players in it."

Conclusion

Overall, setting up ChatGPT API for answering simple queries involving simple SQL statements that GPT must output is relatively simple, but more robust prompt engineering along with other things needs to be done in order to help LLMs learn better how to write correct queries for more complex queries and answer according to a user's question in natural language accurately.