University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2

Date: 2025-10-19

Group Number: 46

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Fiona Zhong	86409539	t8l6n	zhongfiona6@gmail.com
Yichen Li	25684408	e7k2k	liyichen121can@163.com
Junshan Zhou	69637437	m0u7l	junshan_zhou2022@163.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

University of British Columbia, Vancouver
Department of Computer Science

---

2. A brief (~2-3 sentences) summary of your project. Many of your TAs are managing multiple projects so this will help them remember details about your project.

Our project models the core data system for an online multiplayer role-playing game (RPG). The database represents key elements of the virtual world (e.g., players, guilds, pets, items, maps, NPCs, enemies, and missions), allowing the game to track player progress, interactions, and social connections. This structure ensures consistent persistence of gameplay data and supports the interconnected experience typical of modern MMORPGs.

3. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why.

If you have decided not to implement the suggestions given by your project mentor, please be sure to leave a note stating why. This is not to say that you must do everything that your project mentor says. In many instances, there are trade-offs between design choices and your decision may be influenced by different factors. Your TAs will often leave suggestions that are meant to help massage your project into a form that will fit with the requirements in future project milestones. If you choose not to take their advice, it would be helpful for them to know why to better assist the group moving forward

We change our hand-written ER diagram by using draw.io, so that it becomes clearer.

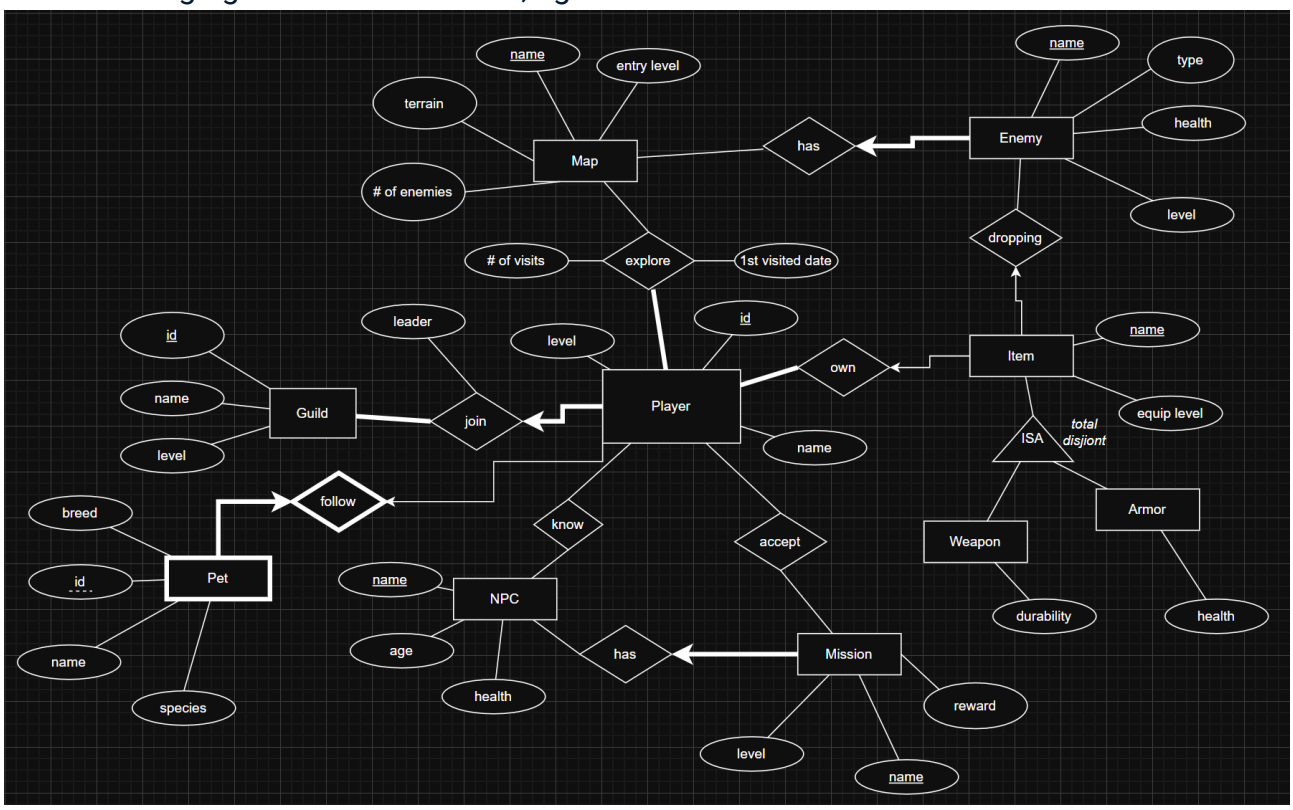
According to the suggestions from our project mentor, We add 2 new attributes (age and health) to entity NPC, because previously NPC only has one attribute npc_name and we set it as primary key, adding more features would make more sense to have a complete and diverse entity. Also it includes another non-PK FD: age -> health.

Additionally, we realize that the leader of the guild should be a player, so we made leader an attribute in the relationship “join” and added a participation constraint in entity Player so that every player must be in a guild. We think we should keep guild_id to serve as a primary key for entity guild, because otherwise it will be impossible to uniquely identify a guild.

In order to meet the requirements of having non-PK/CK FDs for further normalization, we add some attributes to some entities.

1. Adding species to Pet, species -> breed
2. Adding type to Enemy, type&level -> health
3. Adding reward to Mission, level -> reward
4. Adding terrain and # of enemies to Map, entry level -> # of enemies, terrain -> entry level

5. Adding age and health to NPC, age -> health



4. The schema derived from your ER diagram (above).
For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:

a. **List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.**
b. **Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain**

In order to distinguish different names, levels and other attributes in the entities, we use fields like xxx_name, etc., in the schema.

Guild(guild_level: INT, guild_id: INT, guild_name: CHAR(20))

- Primary Key: {guild_id}
- Candidate Key: {guild_id}
- Foreign Key: N/A
- Constraints: guild_level >= 1

join_Player(player_id: INT, player_name: CHAR(20), player_level: INT, leader: CHAR(1), guild_id: INT)

- Primary Key: {player_id}
- Candidate Key: {player_id}
- Foreign Key: guild_id REFERENCES Guild(guild_id)
- Constraints: player_level >= 1, leader: 'Y' or 'N', guild_id NOT NULL

follow_Pet(pet_id: INT, pet_name: CHAR(20), breed: CHAR(20), player_id: INT, species: CHAR(20))

- Primary Key: {pet_id, player_id}
- Candidate Key: {pet_id, player_id}
- Foreign Key: player_id REFERENCES join_Player(player_id)
- Constraints: N/A

Map(map_name: CHAR(20), entry_level: INT, number_of_enemies: INT, terrain: CHAR(20))

- Primary Key: {map_name}
- Candidate Key: {map_name}, {entry_level, terrain, number of enemies}
- Foreign Key: N/A
- Constraints: entry_level >= 1

explore(number_of_visits: INT, first_visited_date: CHAR(20), player_id: INT, map_name: CHAR(20))

- Primary Key: {map_name, player_id}
- Candidate Key: {map_name, player_id}
- Foreign Key: player_id REFERENCES join_Player(player_id)
  map_name REFERENCES Map(map_name)
- Constraints: number_of_visits >= 0

has_Enemy(enemy_name: CHAR(20), enemy_level: INT, enemy_health: INT, map_name: CHAR(20), enemy_type: CHAR(20))

- Primary Key: {enemy_name}
- Candidate Key: {enemy_name}, {enemy_type, enemy_level, enemy_health}
- Foreign Key: map_name REFERENCES Map(map_name)
- Constraints: enemy_level >= 1, enemy_health >= 0, map_name NOT NULL

dropping_own_Items(item_name: CHAR(20), equip_level: INT, enemy_name: CHAR(20), player_id: INT)

- Primary Key: {item_name}
- Candidate Key: {item_name}
- Foreign Key: enemy_name REFERENCES has_Enemy(enemy_name)
  player_id REFERENCE join_Player(player_id)

- Constraint: equip_level >= 1,

Weapon(item_name: CHAR(20), durability: INT)
- Primary Key: {item_name}
- Candidate Key: {item_name}
- Foreign Key: item_name REFERENCES dropping_own_Items(item_name)
- Constraints: (ISA Constraint: Total and disjoint)

Armor(item_name: CHAR(20), health: INT)
- Primary Key: {item_name}
- Candidate Key: {item_name}
- Foreign Key: item_name REFERENCES dropping_own_Items(item_name)
- Constraints: (ISA Constraint: Total and disjoint)

NPC(npc_name: CHAR(20), age: INT, health: INT)
- Primary Key: {npc_name}
- Candidate Key: {npc_name}
- Foreign Key: N/A
- Constraints: age >= 0, health >= 0

know(npc_name: CHAR(20), player_id: INT)
- Primary Key: {npc_name, player_id}
- Candidate Key: {npc_name, player_id}
- Foreign Key: npc_name REFERENCES NPC(npc_name)
  player_id REFERENCES join_Player(player_id)
- Constraints: N/A

has_Mission(mission_name: CHAR(20), mission_level: INT, npc_name: CHAR(20), reward: CHAR(50))
- Primary Key: {mission_name}
- Candidate Key: {mission_name}
- Foreign Key: npc_name REFERENCES NPC(npc_name)
- Constraints: npc_name NOT NULL, mission_level >= 1

accept(player_id: INT, mission_name: CHAR(20))
- Primary Key: {mission_name, player_id}
- Candidate Key: {mission_name, player_id}
- Foreign Key: player_id REFERENCES join_Player(player_id)
  mission_name REFERENCES has_Mission(mission_name)
- Constraints: N/A

5. Functional Dependencies (FDs)

a. **Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key). PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A -> A. Note : In your list of FDs, there must be some kind of valid FD other than those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process. You do not need to have a non-PK/CK FD for each relation but be reasonable. If your TA feels that some non-PK/CK FDs have been omitted, your grade will be adjusted accordingly.**

Guild(guild_level, guild_id, guild_name)
FDs: guild_id -> guild_level, guild_name

join_Player(player_id, player_name, player_level, leader**, guild_id**)
FDs: player_id -> player_name, player_level, guild_leader, guild_name

follow_Pet(pet_id, pet_name, breed, **player_id**, species)
FDs: pet_id, player_id -> pet_name, breed, species
breed -> species

Map(map_name, entry_level, number_of_enemies, terrain)
FDs: map_name -> entry_level, number_of_enemies, terrain
entry_level, number_of enemies, terrain -> map_name
entry level -> number_of_enemies
terrain -> entry level

explore(number_of_visits, first_visited_date, **player_id**, **map_name**)
FDs: player_id, map_name -> number_of_visits, first_visited_date

has_Enemy(enemy_name, enemy_level, enemy_health, **map_name**, enemy_type)
FDs: enemy_name -> enemy_level, enemy_health, map_name, enemy_type
enemy_type, enemy_level, enemy_health -> map_name, enemy_name
enemy_type, enemy_level -> enemy_health

dropping_own_Items(item_name, equip_level, **enemy_name**, **player_id**)
FDs: item_name -> equip_level, enemy_name, play_id

Weapon(**item_name**, durability)
FDs: item_name -> durability

Armor(**item_name**, health)
FDs: item_name -> health

NPC(npc_name, age, health)
FDs: age-> health
npc_name -> age, health

know(**npc_name**, **player_id**)
FDs: npc_name -> player_id
player_id -> npc_name

has_Mission(mission_name, mission_level, **npc_name**, reward)
FDs: mission_name -> mission_level, npc_name, reward
mission_level -> reward

accept(**player_id**, **mission_name**)
FDs: player_id -> mission_name
mission_name -> player_id

---

6. Normalization

a. **Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization.**

You should show the steps taken for the decomposition in a manner similar to that done in class. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown.
The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, ...). ALL Tables must be listed, not only the ones post normalization

We decided to normalize our tables to be in BCNF.

Guild(guild_level, guild_id, guild_name)
FDs: guild_id -> guild_level, guild_name
(Already satisfied BCNF)
Final result:
Guild(guild_level: INT, guild_id: INT, guild_name: CHAR(20))

- Primary Key: {guild_id}
- Candidate Key: {guild_id}
- Foreign Key: N/A
- Constraints: guild_level >= 1

join_Player(player_id, player_name, player_level, leader**, guild_id**)
FDs: player_id -> player_name, player_level, leader, guild_id
(Already satisfied BCNF)
Final result:
join_Player(player_id: INT, player_name: CHAR(20), player_level: INT, leader: CHAR(1), guild_id: INT)

- Primary Key: {player_id}
- Candidate Key: {player_id}
- Foreign Key: guild_id REFERENCES Guild(guild_id)
- Constraints: player_level >= 1, leader: 'Y' or 'N', guild_id NOT NULL

follow_Pet(pet_id, pet_name, breed, **player_id**, species)
FDs: pet_id, player_id -> pet_name, breed, species
(Already satisfied BCNF)

breed -> species
Decompose: follow_Pet1(breed, species)
follow_Pet2(pet_id, pet_name, **breed**, **player_id**)
Final result:
follow_Pet1(breed: CHAR(20), species: CHAR(20))

- Primary Key: {breed}
- Candidate Key: {breed}
- Foreign Key: N/A
- Constraints: N/A

follow_Pet2(pet_id: INT, pet_name: CHAR(20), breed: CHAR(20), player_id: INT)

- Primary Key: {pet_id, player_id}
- Candidate Key: {pet_id, player_id}
- Foreign Key: breed REFERENCES follow_Pet1(breed)
  player_id REFERENCES join_Player(player_id)
- Constraints: N/A

Map(map_name, entry_level, number_of_enemies, terrain)
FDs: map_name -> entry_level, number_of_enemies, terrain
(Already satisfied BCNF)
entry_level, number_of enemies, terrain -> map_name
(Already satisfied BCNF (Candidate Key))
entry level -> number_of_enemies
terrain -> entry level
Decompose: Map1(entry_level, number_of_enemies)
Map'(map_name, **entry_level**, terrain)
Map2(terrain, **entry_level**)
Map3(**terrain**, map_name)
Final result:
Map1(entry_level: INT, number_of_enemies: INT)

- Primary Key: {entry_level}
- Candidate Key: {entry_level}
- Foreign Key: N/A
- Constraints: entry_level >= 1

Map2(entry_level: INT, terrain: CHAR(20))

- Primary Key: {terrain}

- Candidate Key: {terrain}
- Foreign Key: entry_level REFERENCES Map1(entry_level)
- Constraints: entry_level >= 1

Map3(map_name: CHAR(20), terrain: CHAR(20))
- Primary Key: {map_name}
- Candidate Key: {map_name}
- Foreign Key: terrain REFERENCES Map2(terrain)
- Constraints: N/A

explore(number_of_visits, first_visited_date, **player_id**, **map_name**)
FDs: player_id, map_name -> number_of_visits, first_visited_date
(Already satisfied BCNF)
Final result:
explore(number_of_visits: INT, first_visited_date: CHAR(20), player_id: INT, map_name: CHAR(20))
- Primary Key: {map_name, player_id}
- Candidate Key: {map_name, player_id}
- Foreign Key: player_id REFERENCES join_Player(player_id)
  map_name REFERENCES Map3(map_name)
- Constraints: number_of_visits >= 0

has_Enemy(enemy_name, enemy_level, enemy_health, **map_name**, enemy_type)
FDs: enemy_name -> enemy_level, enemy_health, map_name, enemy_type
(Already satisfied BCNF)
enemy_type, enemy_level, enemy_health -> map_name, enemy_name
(Already satisfied BCNF (Candidate Key))
enemy_type, enemy_level -> enemy_health
Decompose: has_enemy1(enemy_type, enemy_level, enemy_health)
has_enemy2(enemy_name, **enemy_level**, **map_name**, **enemy_type**)
Final Result:
has_Enemy1(enemy_level: INT, enemy_health: INT, enemy_type: CHAR(20))
- Primary Key: {enemy_type, enemy_level}
- Candidate Key: {enemy_type, enemy_level}
- Foreign Key: N/A

- Constraints: enemy_level >= 1, enemy_health >= 0

has_Enemy2(enemy_name: CHAR(20), enemy_level: INT, map_name: CHAR(20), enemy_type: CHAR(20))

- Primary Key: {enemy_name}
- Candidate Key: {enemy_name}
- Foreign Key: map_name REFERENCES Map3(map_name)
  enemy_level REFERENCES has_enemy1(enemy_level)
  enemy_type REFERENCES has_enemy1(enemy_type)
- Constraints: enemy_level >= 1 NOT NULL, map_name NOT NULL, enemy_type NOT NULL

dropping_own_Items(item_name, equip_level, **enemy_name**, **player_id**)
FDs: item_name -> equip_level, enemy_name, player_id
(Already in BCNF)
Final result:
dropping_own_Items(item_name: CHAR(20), equip_level: INT, enemy_name: CHAR(20), player_id: INT)

- Primary Key: {item_name}
- Candidate Key: {item_name}
- Foreign Key: enemy_name REFERENCES has_Enemy(enemy_name)
  player_id REFERENCE join_Player(player_id)
- Constraint: equip_level >= 1

Weapon(**item_name**, durability)
FDs: item_name -> durability
(Already in BCNF)
Final result:
Weapon(item_name: CHAR(20), durability: INT)

- Primary Key: {item_name}
- Candidate Key: {item_name}
- Foreign Key: item_name REFERENCES dropping_own_Items(item_name)
- Constraints: (ISA Constraint: Total and disjoint)

Armor(**item_name**, health)
FDs: item_name -> health
(Already in BCNF)
Final result:

Armor(item_name: CHAR(20), durability: INT)

- Primary Key: {item_name}
- Candidate Key: {item_name}
- Foreign Key: item_name REFERENCES dropping_own_Items(item_name)
- Constraints: (ISA Constraint: Total and disjoint)

NPC(<u>npc_name</u>, age, health)
FDs: age-> health (not in BCNF)
npc_name -> age, health
(Already in BCNF)

Decompose: NPC1(<u>age</u>, health)
NPC2(<u>npc_name</u>, **age**)
Final Result:
NPC1(age: INT, health: INT)

- Primary Key:{age}
- Candidate Key: {age}
- Foreign Key: N/A
- Constraints: age >= 1, health >= 1

NPC2(npc_name: CHAR(20), age: INT)

- Primary Key:{npc_name}
- Candidate Key: {npc_name}
- Foreign Key: age REFERENCES NPC1(age)
- Constraints: age >= 1 NOT NULL

know(**<u>npc_name</u>**, **<u>player_id</u>**)
FDs: npc_name -> player_id
(Already in BCNF)
player_id -> npc_name
(Already in BCNF)
Final result:
know(npc_name: CHAR(20), player_id: INT)

- Primary Key: {npc_name, player_id}
- Candidate Key: {npc_name, player_id}
- Foreign Key: npc_name REFERENCES NPC2(npc_name)

player_id REFERENCES join_Player(player_id)

- Constraints: N/A

has_Mission(mission_name, mission_level, **npc_name**, reward)
FDs: mission_name -> mission_level, npc_name, reward
(Already in BCNF)
mission_level -> reward
Decompose: has_Mission1(mission_level, reward)
has_Mission2(mission_name, **mission_level**, **npc_name**)
Final Result:
has_Mission1(mission_level: INT, reward: CHAR(50))

- Primary Key:{mission_level}
- Candidate Key: {mission_level}
- Foreign Key: N/A
- Constraints: mission_level >= 0

has_Mission2(mission_name: CHAR(20), mission_level: INT, npc_name: CHAR(20))

- Primary Key:{mission_name}
- Candidate Key: {mission_name}
- Foreign Key: npc_name REFERENCES NPC(npc_name)
  mission_level REFERENCES has_Mission1(mission_level)
- Constraints: npc_name NOT NULL, mission_level NOT NULL

accept(**player_id**, **mission_name**)
FDs: player_id -> mission_name
(Already in BCNF)
mission_name -> player_id
(Already in BCNF)
Final result:
accept(player_id: INT, mission_name: CHAR(20))

- Primary Key: {mission_name, player_id}
- Candidate Key: {mission_name, player_id}
- Foreign Key: mission_name REFERENCES has_Mission2(mission_name)
  player_id REFERENCES join_Player(player_id)
- Constraints: N/A

7. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc.
Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a CHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use CHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.

```
CREATE TABLE Guild(
   guild_level INTEGER,
   guild_id INT,
   guild_name VARCHAR(20),
   PRIMARY KEY (guild_id)
);

CREATE TABLE join_Player(
        player_id INTEGER PRIMARY KEY,
        player_name VARCHAR(20),
        player_level INTEGER CHECK (player_level >= 1),
        leader CHAR(1) CHECK (leader IN ('Y','N')),
        guild_id INT NOT NULL,
        FOREIGN KEY (guild_id) REFERENCES Guild(guild_id)
        ON DELETE CASCADE
);
```

Reasoning:
If a guild is deleted, the players should be deleted — they cannot continue to exist without belonging to a guild.

```
CREATE TABLE follow_Pet1(
   breed   VARCHAR(20) PRIMARY KEY,
   species VARCHAR(20)
);

CREATE TABLE follow_Pet2(
   pet_id     INTEGER,
```

```
  pet_name   VARCHAR(20),
  breed      VARCHAR(20),
  player_id  INTEGER NOT NULL,
  PRIMARY KEY (pet_id, player_id),
  UNIQUE (player_id),
  FOREIGN KEY (breed) REFERENCES follow_Pet1(breed)
    ON DELETE CASCADE,
  FOREIGN KEY (player_id) REFERENCES join_Player(player_id)
    ON DELETE CASCADE
);
```

Reasoning:

ON DELETE CASCADE on breed:
If a pet breed is removed from the database, all pets of that breed should also be deleted.

ON DELETE CASCADE on player_id:
If a player is deleted, all their pets disappear with them since they no longer exist in the game world.

```
CREATE TABLE Map1(
  entry_level      INTEGER PRIMARY KEY,
  Number_of_enemies INTEGER,
  CHECK (entry_level >= 1)
);

CREATE TABLE Map2(
  terrain     VARCHAR(20) PRIMARY KEY,
  entry_level INTEGER NOT NULL CHECK (entry_level >= 1),
  FOREIGN KEY (entry_level) REFERENCES Map1(entry_level)
    ON DELETE CASCADE
);
```

Reasoning:
If a map level is deleted from Map1, all terrains (Map2) associated with that level should also be removed. Otherwise, the terrain would reference a non-existent entry level.

```
CREATE TABLE Map3(
```

```
  map_name VARCHAR(20) PRIMARY KEY,
  terrain  VARCHAR(20),
  FOREIGN KEY (terrain) REFERENCES Map2(terrain)
    ON DELETE CASCADE
);
```

Reasoning:
If a terrain is deleted, all maps that use that terrain should also be deleted.

```
CREATE TABLE explore(
  number_of_visits   INTEGER CHECK (number_of_visits >= 0),
  first_visited_date VARCHAR(20),
  player_id          INTEGER,
  map_name           VARCHAR(20),
  PRIMARY KEY (map_name, player_id),
  FOREIGN KEY (player_id) REFERENCES join_Player(player_id)
    ON DELETE CASCADE,
  FOREIGN KEY (map_name) REFERENCES Map3(map_name)
    ON DELETE CASCADE
);
```

Reasoning:

ON DELETE CASCADE on player_id:

If a player is deleted, their exploration records vanish too.

ON DELETE CASCADE on map_name:

If a map is deleted, the exploration logs for that map are removed — you can't explore a map that no longer exists.

```
CREATE TABLE has_enemy1(
  enemy_type  VARCHAR(20),
  enemy_health INTEGER CHECK (enemy_health >= 0),
  enemy_level  INTEGER CHECK (enemy_level >= 1),
  PRIMARY KEY (enemy_type, enemy_level)
);
```

```
CREATE TABLE has_enemy2(
```

```
  enemy_name  VARCHAR(20) PRIMARY KEY,
  enemy_level INTEGER CHECK (enemy_level >= 1) NOT NULL,
  map_name    VARCHAR(20) NOT NULL,
  enemy_type  VARCHAR(20) NOT NULL,
  FOREIGN KEY (map_name) REFERENCES Map3(map_name)
    ON DELETE CASCADE,
  FOREIGN KEY (enemy_type, enemy_level)
    REFERENCES has_enemy1(enemy_type, enemy_level)
    ON DELETE CASCADE
);
```

Reasoning:

ON DELETE CASCADE on map_name:

If a map is deleted, all enemies on that map should be deleted (since they belong to that location).

ON DELETE CASCADE on enemy_type, enemy_level:

If a type of enemy (and its level) is deleted from the master list, all specific instances of that enemy type are deleted for consistency.

```
CREATE TABLE dropping_own_Items (
  item_name  VARCHAR(20) PRIMARY KEY,
  equip_level INTEGER,
  enemy_name VARCHAR(20),
  player_id  INTEGER,
  FOREIGN KEY (enemy_name) REFERENCES has_enemy2(enemy_name)
    ON DELETE SET NULL,
  FOREIGN KEY (player_id) REFERENCES join_Player(player_id)
    ON DELETE SET NULL
);
```

Reasoning:

ON DELETE SET NULL on enemy_name:

If an enemy is deleted, its dropped items might still exist in the world but no longer have a known origin — enemy_name becomes NULL.

ON DELETE SET NULL on player_id:

If a player is deleted, the item can remain in the game world, but its owner is now NULL (unowned).
This preserves items even if players or enemies are gone.

```
CREATE TABLE Weapon (
   item_name  VARCHAR(20) PRIMARY KEY,
   durability INTEGER,
   FOREIGN KEY (item_name) REFERENCES dropping_own_Items(item_name)
      ON DELETE CASCADE
);
```

Reasoning:
If a base item (in dropping_own_Items) is deleted, the weapon must also be deleted — otherwise, it would reference a non-existent item.

```
CREATE TABLE Armor (
   item_name VARCHAR(20) PRIMARY KEY,
   health    INTEGER,
   FOREIGN KEY (item_name) REFERENCES dropping_own_Items(item_name)
      ON DELETE CASCADE
);
```

Reasoning:
If a base item (in dropping_own_Items) is deleted, the armor must also be deleted — otherwise, it would reference a non-existent item.

```
CREATE TABLE NPC1 (
   age    INTEGER PRIMARY KEY CHECK (age >= 1),
   health INTEGER CHECK (health >= 1)
);

CREATE TABLE NPC2 (
   npc_name VARCHAR(20) PRIMARY KEY,
   age      INTEGER NOT NULL CHECK (age >= 1),
   FOREIGN KEY (age) REFERENCES NPC1(age)
      ON DELETE CASCADE
);
```

Reasoning:
If a base NPC age profile is deleted, all NPCs with that profile should also be deleted.

```
CREATE TABLE know (
   npc_name  VARCHAR(20),
   player_id INTEGER,
   PRIMARY KEY (npc_name, player_id),
   FOREIGN KEY (npc_name) REFERENCES NPC2(npc_name)
      ON DELETE CASCADE,
   FOREIGN KEY (player_id) REFERENCES join_Player(player_id)
      ON DELETE CASCADE
);
```

Reasoning:
If either the NPC or the player is deleted, their “knowledge” or relationship record should disappear as well.

```
CREATE TABLE has_Mission1 (
   mission_level INTEGER PRIMARY KEY CHECK (mission_level >= 0),
   reward       VARCHAR(50)
);

CREATE TABLE has_Mission2 (
   mission_name  VARCHAR(20) PRIMARY KEY,
   mission_level INTEGER NOT NULL,
   npc_name      VARCHAR(20) NOT NULL,
   FOREIGN KEY (mission_level) REFERENCES has_Mission1(mission_level)
      ON DELETE CASCADE,
   FOREIGN KEY (npc_name) REFERENCES NPC2(npc_name)
      ON DELETE CASCADE
);
```

Reasoning:

ON DELETE CASCADE on mission_level:

If a mission level is removed, all missions at that level should go too.

ON DELETE CASCADE on npc_name:

If an NPC is deleted, all missions associated with that NPC must also be deleted — they no longer exist to assign the mission.

```
CREATE TABLE accept (
   mission_name VARCHAR(20),
   player_id    INTEGER,
   PRIMARY KEY (mission_name, player_id),
   FOREIGN KEY (mission_name) REFERENCES has_Mission2(mission_name)
      ON DELETE CASCADE,
   FOREIGN KEY (player_id) REFERENCES join_Player(player_id)
      ON DELETE CASCADE
);
```

Reasoning:

ON DELETE CASCADE on mission_name:

If a mission is deleted, all acceptance records must be deleted.

ON DELETE CASCADE on player_id:

If a player is deleted, their accepted missions must also be removed.

8. INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later.

```
insert into Guild values (2, 1, 'First');
insert into Guild values (4, 2, 'Second');
insert into Guild values (5, 3, 'Third');
insert into Guild values (5, 4, 'Fourth');
insert into Guild values (6, 5, 'Fifth');


insert into join_Player values (1, 'Alice', 1, 'Y', 1);
insert into join_Player values (2, 'Bob', 2, 'N', 1);
insert into join_Player values (3, 'David', 3, 'Y', 2);
insert into join_Player values (4, 'Jonny', 4, 'Y', 2);
insert into join_Player values (5, 'Kitty', 3, 'N', 3);

insert into follow_Pet1 values ('DireWolf', 'Wolf');
insert into follow_Pet1 values ('FireHawk', 'Bird');
```

```
insert into follow_Pet1 values ('ShadowCat', 'Feline');
insert into follow_Pet1 values ('IronBoar', 'Boar');
insert into follow_Pet1 values ('FrostBear', 'Bear');
insert into follow_Pet1 values ('StoneDog', 'Dog');


insert into follow_Pet2 values (1, 'Lupus',   'DireWolf', 1);
insert into follow_Pet2 values (2, 'Blaze',   'FireHawk', 2);
insert into follow_Pet2 values (3, 'Saber',   'ShadowCat',3);
insert into follow_Pet2 values (4, 'Tusk',    'IronBoar', 4);
insert into follow_Pet2 values (5, 'IcePaw',  'FrostBear',5);


insert into Map1 values (1, 10);
insert into Map1 values (5, 15);
insert into Map1 values (10, 20);
insert into Map1 values (15, 25);
insert into Map1 values (20, 30);

insert into Map2 values ('Forest',   1);
insert into Map2 values ('Desert',   5);
insert into Map2 values ('Mountain', 10);
insert into Map2 values ('Swamp',    15);
insert into Map2 values ('Volcano',  20);

insert into Map3 values ('Greenwood',    'Forest');
insert into Map3 values ('Sands of Time','Desert');
insert into Map3 values ('Iron Peaks',   'Mountain');
insert into Map3 values ('Murkfen',      'Swamp');
insert into Map3 values ('Lava Fields',  'Volcano');


insert into explore values (5, '2025-01-15', 1, 'Greenwood');
insert into explore values (3, '2025-02-10', 2, 'Sands of Time');
insert into explore values (7, '2025-03-05', 3, 'Iron Peaks');
insert into explore values (2, '2025-04-12', 4, 'Murkfen');
insert into explore values (4, '2025-05-20', 5, 'Lava Fields');


insert into has_enemy1 values ('Goblin', 50,  1);
```

```
insert into has_enemy1 values ('Troll',  100, 5);
insert into has_enemy1 values ('Orc',    150, 10);
insert into has_enemy1 values ('Dragon', 200, 15);
insert into has_enemy1 values ('Demon',  250, 20);


insert into has_enemy2 values ('Gobbo',      1,  'Greenwood',    'Goblin');
insert into has_enemy2 values ('TrollKing',  5,  'Sands of Time','Troll');
insert into has_enemy2 values ('OrcWarlord', 10, 'Iron Peaks',   'Orc');
insert into has_enemy2 values ('FireDrake',  15, 'Murkfen',      'Dragon');
insert into has_enemy2 values ('Hellspawn',  20, 'Lava Fields',  'Demon');


insert into dropping_own_Items values ('IronSword',  5,  'Gobbo',     1);
insert into dropping_own_Items values ('FireStaff', 10,  'TrollKing',  2);
insert into dropping_own_Items values ('WarAxe',    15,  'OrcWarlord', 3);
insert into dropping_own_Items values ('DragonBow', 20,  'FireDrake',  4);
insert into dropping_own_Items values ('DemonBlade',25,  'Hellspawn',  5);


insert into dropping_own_Items values ('LeatherArmor',  1,  null, null);
insert into dropping_own_Items values ('ChainMail',     5,  null, null);
insert into dropping_own_Items values ('PlateArmor',    10, null, null);
insert into dropping_own_Items values ('DragonScale',   15, null, null);
insert into dropping_own_Items values ('DemonHide',     20, null, null);

insert into Weapon values ('IronSword',  100);
insert into Weapon values ('FireStaff',  80);
insert into Weapon values ('WarAxe',     90);
insert into Weapon values ('DragonBow',  70);
insert into Weapon values ('DemonBlade', 95);

insert into Armor values ('LeatherArmor', 50);
insert into Armor values ('ChainMail',    75);
insert into Armor values ('PlateArmor',   100);
insert into Armor values ('DragonScale',  125);
insert into Armor values ('DemonHide',    150);

insert into NPC1 values (20, 100);
insert into NPC1 values (30, 120);
```

```
insert into NPC1 values (40, 140);
insert into NPC1 values (50, 160);
insert into NPC1 values (60, 180);

insert into NPC2 values ('ElderSage',   60);
insert into NPC2 values ('Blacksmith',  40);
insert into NPC2 values ('Merchant',    30);
insert into NPC2 values ('GuardCaptain',50);
insert into NPC2 values ('Healer',      20);

insert into know values ('ElderSage',   1);
insert into know values ('Blacksmith',  2);
insert into know values ('Blacksmith',  3);
insert into know values ('Merchant',    3);
insert into know values ('GuardCaptain',4);
insert into know values ('Healer',      1);

insert into has_Mission1 values (1,  '100 Gold');
insert into has_Mission1 values (5,  'Magic Gem');
insert into has_Mission1 values (10, 'Enchanted Sword');
insert into has_Mission1 values (15, 'Dragon Scale');
insert into has_Mission1 values (20, 'Demon Heart');

insert into has_Mission2 values ('SlayGoblins',  1,  'ElderSage');
insert into has_Mission2 values ('DesertQuest',  5,  'Merchant');
insert into has_Mission2 values ('PeakChallenge',10, 'Blacksmith');
insert into has_Mission2 values ('SwampHunt',    15, 'GuardCaptain');
insert into has_Mission2 values ('VolcanoTrial', 20, 'Healer');

insert into accept values ('SlayGoblins',  1);
insert into accept values ('DesertQuest',  2);
insert into accept values ('PeakChallenge',2);
insert into accept values ('PeakChallenge',3);
insert into accept values ('SwampHunt',    4);
insert into accept values ('VolcanoTrial', 5);
```

9. An explicit acknowledgment about your use of AI tools in this assignment. Specifically, we are looking for a clear yes/no about whether you have used one or more AI tools. If

yes, we want to know which tool(s) you have used and we want a PDF of the full conversation.

In this project, we did not use AI tools to gain support for any purpose.

Check the milestone 2 assignment on Canvas for the grading rubric. Refer to the syllabus for information on late submission/penalty rules.

Notes

1. Be consistent with the names used in your ER diagram, schema, and FDs. Make a note if the name has been intentionally changed.
2. As you start analyzing these requirements, you may notice that certain details are missing. In this case, you may make any reasonable assumptions about them; but, if there is any uncertainty about some requirements, you should ask your project TA before proceeding further (or if it's more general in nature, post your question on Piazza).
   Furthermore, it is acceptable to modify your design from your original project proposal, because as you progress and start thinking more about the data and the queries that you want to answer from your application, it is normal to find that you need to modify the design (but don't go back and re-do or re-submit your project proposal)

3. If you want to start implementation, start implementing your front end.
   Don’t work on the back end yet as your TA may have some feedback for you that will result in backend changes

You need to use the repository that the course has provided for you. Be sure to double check your Git config options to ensure that you are making commits under your own name.