Use Case Study Report

Andrew Fox and Chien Nguyen

SUMMARY

The objective of this study was to propose, design, model, and implement a relational database fit for a particular business application. We conceptualized a business that hosts commercial treasure hunts for cash prizes. Such treasure hunts would be facilitated primarily through a mobile or web application and operate almost entirely without employees. Considering the nature of the business model, a relational database proved fit to store and manage the constantly changing user, treasure, and financial data.

The presence of adventure and navigation apps in the mobile gaming scene is nothing new, with the category having grossed two billion U.S. dollars in 2022 according to Statistica.com. Treasure hunting games facilitated through mobile apps like geocaching see hundreds of thousands of users hiding and finding caches each year. By way of a financial incentive, our application would operate self-sufficiently and encourage users anywhere in the world to create and participate in local treasure hunts, encouraging natural exploration, problem solving, and teamwork.

In what follows we outline the general proposal and database requirements of the application.

Every treasure hunt has a small entry fee, which is pooled as the cash prize for the hunt and rewarded to the hunter who discovers it. A cut of the prize is taken by the company for business expenses and profit, and another small portion is rewarded to the user who created the hunt. This incentivizes users to not only participate in hunts but to create them as well, ensuring a consistent availability of treasure hunts.

Users who create treasure hunts cannot participate in any hunts while their hunt is active. The location of the treasure, as indicated by geographic coordinates, is generated randomly and kept secret from the creator to prevent cheating. A series of clues are generated by tailored artificial intelligence programs and based on the surrounding geography, landmarks, communities, etc., so as to help guide hunters to the location. Each treasure hunt has a time limit, at the end of which the entry fees are refunded. As time progresses, more and more clues will be revealed to the hunters. When a hunter is within a certain radius of the generated coordinates, the app would consider them as having found the treasure.

For each individual hunt, a unique hunt ID, the geographic latitude and longitude, the entry fee, the total cash prize, a difficulty rating, the creation date, the discovery date, the time limit and completion date, the number of participating hunters, the user who created the hunt, and the user who found the treasure, all need to be recorded. Before a hunt can commence, a minimum number of hunters need to sign up to ensure a significant payout and competition. This minimum number of participating hunters must also be recorded.

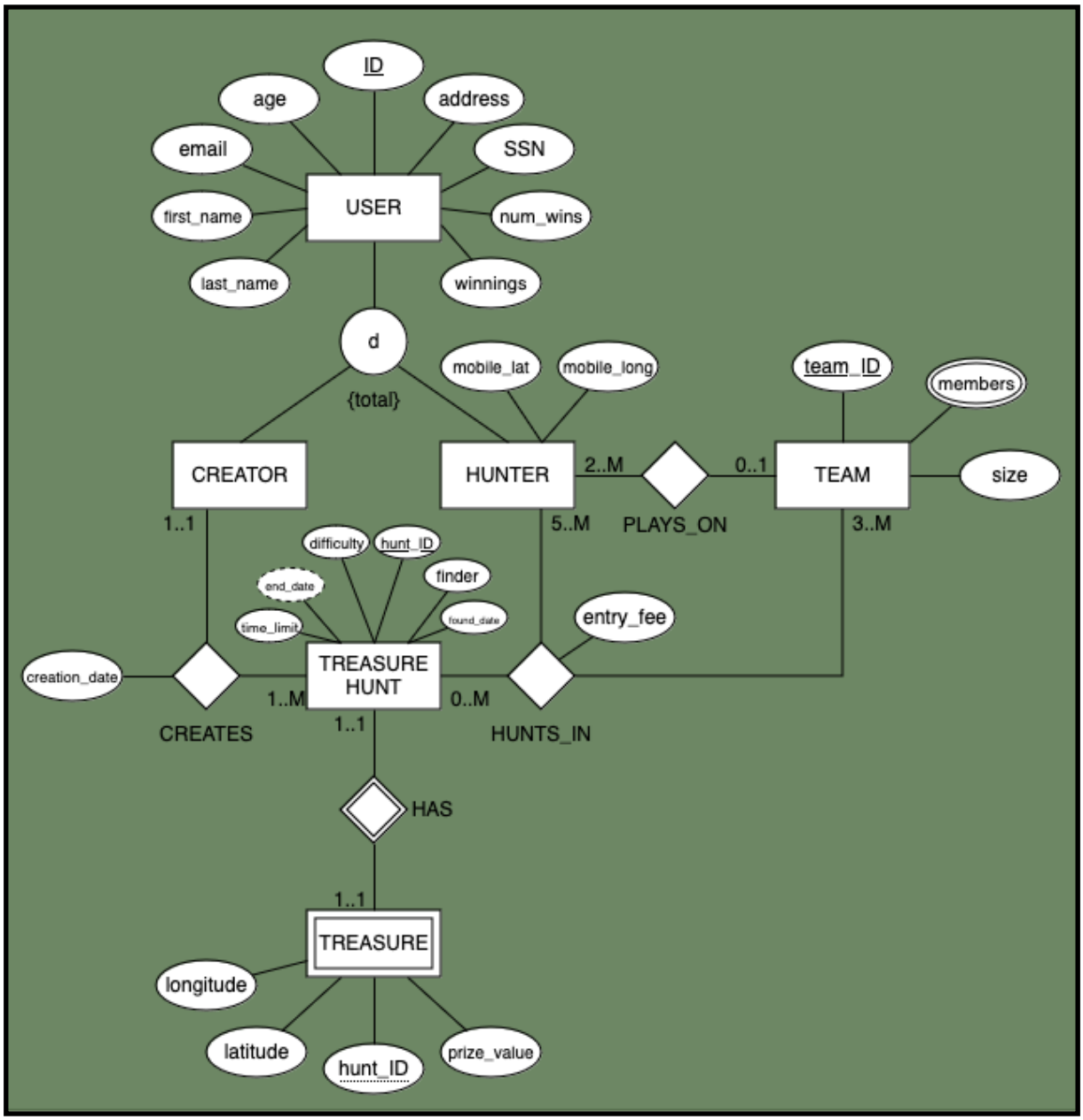
A user must either be a hunter or a creator, but cannot be both at the same time. A creator cannot participate in any hunts, especially any they've created, at any time while they are registered as a creator. A creator can switch to a hunter as long as no hunt they've created is active. A hunter can switch to a creator as long as they are not actively participating in a hunt. Users registered as hunters can participate in two hunts at a time. All users must be 21 years of age. Users can form and join teams in which they would participate in the hunt together. For these reasons, a unique user ID, SSN, first name, last name, age, address, email, their total wins, and their total winnings all need to be recorded in the database. The geographic coordinates of their mobile device must be recorded only for Hunters. The members of a team, the team size, and a team ID would all be recorded for teams.

Below we outline the relationship requirements that would be present in a relational model.

- A hunter can enter 0 to infinite treasure hunts, but can only actively participate in at most 2 hunts. Treasure hunts must be entered by at least 5 or more hunters before commencing. Hunters cannot be creators.
- Hunters can join 0 or 1 team. A team must have at least 2 and up to as many Hunters. Any hunter in a team must share the cash prize of treasures that team finds.
- A creator must create 1 and up to an infinite number of treasure hunts, but the location is created randomly by software. A treasure hunt can be created by 1 and only 1 creator. Creators cannot be hunters while their created hunts are active.
- Every treasure hunt has one and only one treasure.

CONCEPTUAL MODELLING

EER Diagram



Overview of EER Model

USER is an entity with a disjoint, total specialization into **CREATOR** and **HUNTER** entities. This is because a user must be either a creator or a hunter, but cannot be both at the same time. ID is the primary key of **USER**. Only the longitude and latitude is recorded for **HUNTERS**

One and only one **CREATOR** can *create* at least one and up to many **TREASURE HUNTS**. At least five and up to many **HUNTERS** may *hunt in* zero or many **TREASURE HUNTS**.

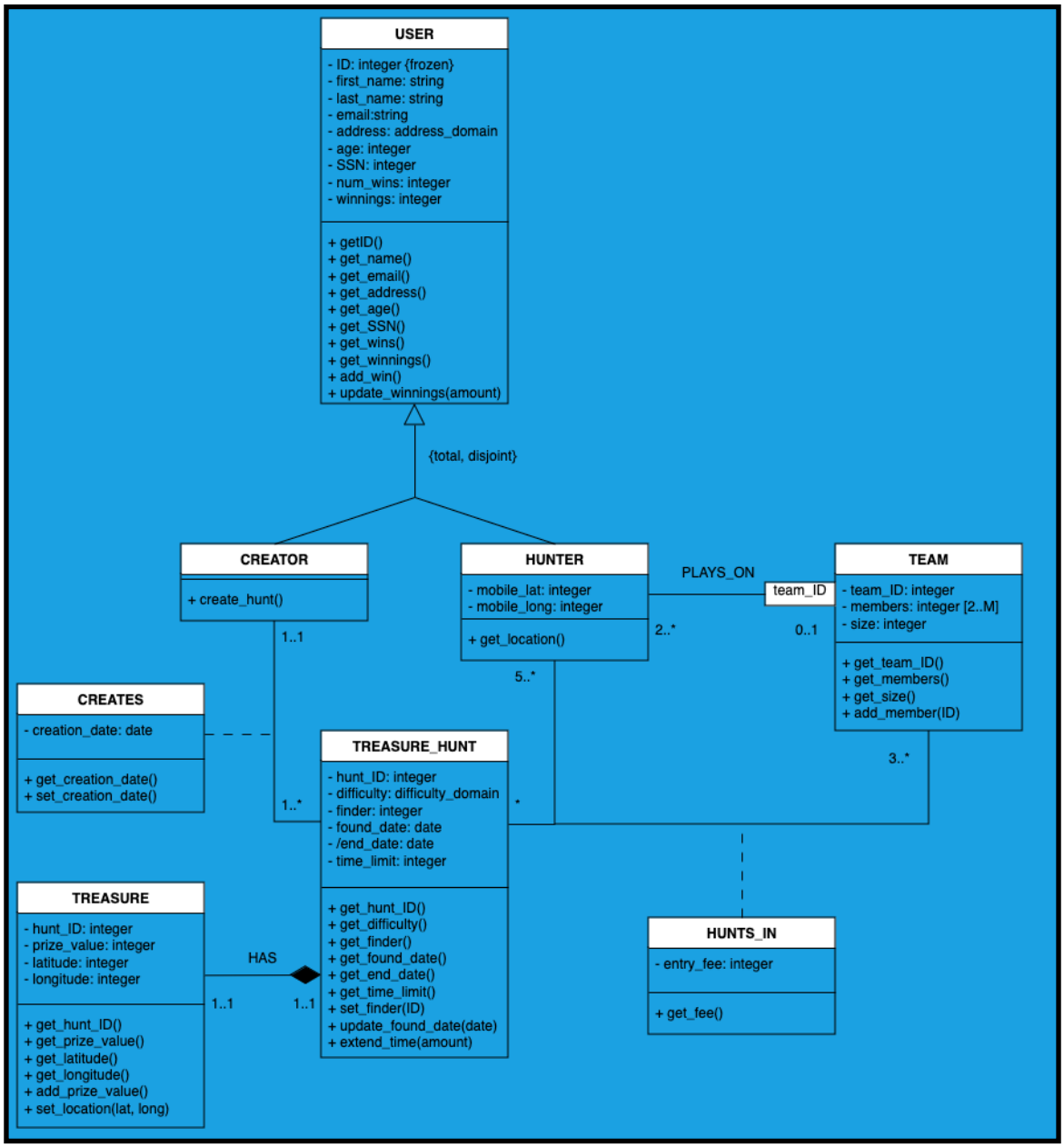
At least 2 and up to many **HUNTERS** can *play on* zero or many **TEAMS**. The team_ID is the primary key for a **TEAM**. The members of a **TEAM** is a multi-valued attribute type, since multiple members may play on the team. At least 5 or many **HUNTERS** and/or at least 3 or many **TEAMS** can *hunt in* zero or many **TREASURE HUNTS**.

A **TREASURE HUNT** *has* one and only one **TREASURE**, which is existence-dependent on the former. The end_date for a **TREASURE HUNT** is a derived variable type since it can be calculated from the found_date or time_limit. The primary key of **TREASURE HUNT** is it's hunt_ID. **TREASURE** has a foreign key referring to this hunt_ID in **TREASURE HUNT**.

Of course, the EER model has several limitations.

- The model demonstrates the disjointness of the specialization at any given time, but not across time. That is, it cannot model the fact that Users can switch between being a Hunter or Creator under certain conditions, those being whether or not they are hunting in or have created active hunts. It can only model that at any given time, a User can only be a Hunter or a Creator.
- There is no support for the definition of functions, which would be incredibly useful in several scenarios. For instance, when new hunters enter a Treasure Hunt, the prize_value of the Treasure should be updated according to the entry_fee. Another very useful function would be the generation of a treasure's random location upon creation. Other examples include updating the size of a team when hunters join it or updating the finder, found_date, end_date, num_wins, and total_winnings attributes when a treasure is found.
- The domains of attributes cannot be defined in the EER model. While some domains are easily identifiable, it could be beneficial to know some others. The various IDs, for example, could have different integer domains. The difficulty attribute of a Treasure Hunt may even have a unique domain.

UML Diagram


USER
- ID: integer {frozen}
- first_name: string
- last_name: string
- email:string
- address: address_domain
- age: integer
- SSN: integer
- num_wins: integer
- winnings: integer
+ getID()
+ get_name()
+ get_email()
+ get_address()
+ get_age()
+ get_SSN()
+ get_wins()
+ get_winnings()
+ add_win()
+ update_winnings(amount)
{total, disjoint}
CREATOR
+ create_hunt()
HUNTER
- mobile_lat: integer
- mobile_long: integer
+ get_location()
PLAYS_ON
team_ID
2..*
0..1
TEAM
- team_ID: integer
- members: integer [2..M]
- size: integer
+ get_team_ID()
+ get_members()
+ get_size()
+ add_member(ID)
1..1
5..*
3..*
CREATES
- creation_date: date
+ get_creation_date()
+ set_creation_date()
1..*
*
TREASURE_HUNT
- hunt_ID: integer
- difficulty: difficulty_domain
- finder: integer
- found_date: date
- /end_date: date
- time_limit: integer
+ get_hunt_ID()
+ get_difficulty()
+ get_finder()
+ get_found_date()
+ get_end_date()
+ get_time_limit()
+ set_finder(ID)
+ update_found_date(date)
+ extend_time(amount)
TREASURE
- hunt_ID: integer
- prize_value: integer
- latitude: integer
- longitude: integer
+ get_hunt_ID()
+ get_prize_value()
+ get_latitude()
+ get_longitude()
+ add_prize_value()
+ set_location(lat, long)
HAS
1..1
1..1
HUNTS_IN
- entry_fee: integer
+ get_fee()

Overview of UML Diagram

Everything from the EER model just about carries over. The UML diagram is able to model certain features differently. There is a qualified association between **TEAM** and **HUNTER**. There is a composite aggregation between **TREASURE** and **TREASURE_HUNT**, as only one treasure can belong to a hunt.

An advantage of the UML diagram is its ability to model functions. Several important functions are modeled here. There is an update_winnings() and an add_win() function for **USERS** to update their information regarding treasure hunts. **CREATORS** have a create_hunt() function. **TEAMS** has an add_member() function to add hunters if they join the team. **TREASURE** has a set_location(lat, long) function which would randomly set the location of the treasure, as well as a add-prize_value(amount) function to update the prize value as hunters sign up for the hunt. **TREASURE_HUNT** has several functions to update the finder if and when it is found, as well as a extend_time() function to adjust the time limit of the hunt.

Also of note is the **USER** ID variable, which has a frozen changeability property assigned to it. This is because the ID is a unique identifier and once a user has an ID it should not change.

MAPPING CONCEPTUAL MODEL TO LOGICAL MODEL

With the conceptual EER and UML models designed, the database can be mapped to a logical model. In this case, we mapped it to a relational model.

USERS (ID, SSN, first_name, last_name, email, age, address, num_wins, winnings)

CREATOR (*C_ID*)
- FK *C_ID* refers to PK ID in **USERS;** NOT NULL.

TEAM (team_ID, size)

HUNTER (*H_ID*, mobile_lat, mobile_long, *team_ID*)
- FK *H_ID* refers to PK ID in **USERS;** NOT NULL.
- FK *team_ID* refers to PK team_ID in **TEAM;** NULL ALLOWED.

MEMBERS (*team_ID, H_ID*)
- FK *team_ID* refers to PK team_ID in **TEAM;** NOT NULL.
- FK *H_ID* refers to PK *H_ID* in **HUNTER;** NOT NULL.

TREASURE_HUNT (hunt_ID, difficulty, *finder*, found_date, end_date, time_limit, creation_date, *C_ID*)
- FK *C_ID* refers to PK *C_ID* in **CREATOR;** NOT NULL.
- FK *finder* refers to PK ID in **USERS;** NULL ALLOWED.

HUNTS_IN (*H_ID*, *team_ID*, *hunt_ID*, entry_fee)
- FK *H_ID* refers to PK *H_ID* in **HUNTER;** NOT NULL.
- FK *team_ID* refers to PK team_ID in **TEAM;** NULL ALLOWED.
- FK *hunt_ID* refers to PK hunt_ID in **TREASURE_HUNT;** NOT NULL.

TREASURE (*hunt_ID*, prize_value, latitude, longitude)
- FK *hunt_ID* refers to PK hunt_ID in **TREASURE_HUNT;** NOT NULL.

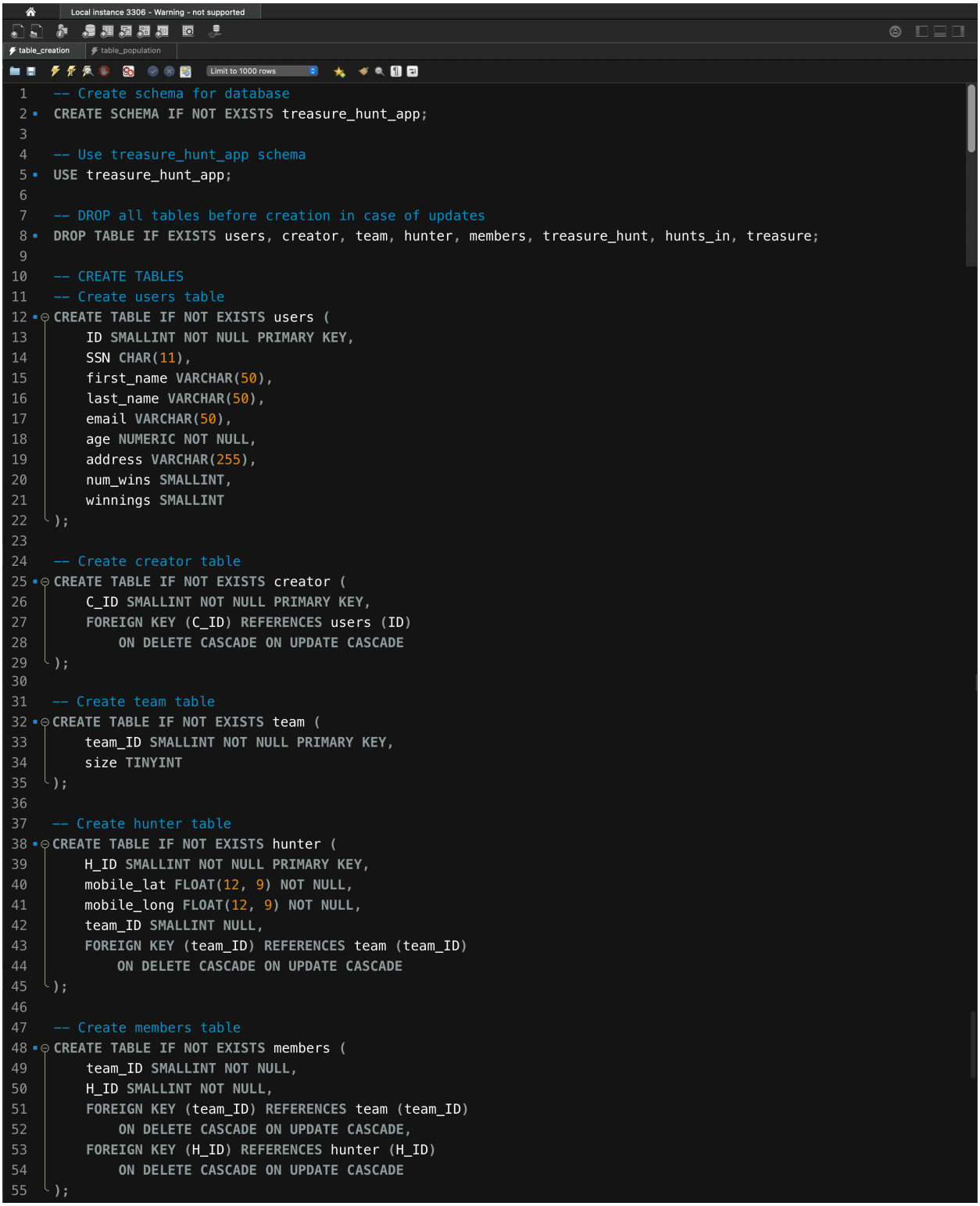
Summary of Mapping & Semantics Lost

- Every entity type was mapped to its own relation, with their simple attribute types being directly mapped to columns and their key attribute type being made a primary key.
- To map the total, disjoint specialization of **USER** into **CREATOR** and **HUNTER**, we chose to map both **CREATOR** and **HUNTER** to their own entity types. Each is linked to the **USER** relation through a primary/foreign key referring to ID in **USER**. While this isn't ideal for total, disjoint specializations, this option facilitated best the relationship types they have.
  - The disjoint nature cannot be enforced; that is, the relational model cannot enforce that a user cannot be a creator and a hunter simultaneously.
- The members attribute of **TEAM** is multi-valued, so it had to be extracted into its own relation, **MEMBERS**. The foreign key of **MEMBERS** refers to team_ID in **TEAM**.
- **TREASURE** is existent-dependent on **TREASURE_HUNT** in a one-to-one relation, so hunt_ID was included as a foreign key in **TREASURE** referring to hunt_ID in **TREASURE_HUNT**.
  - All cardinalities are supported here.
- For the one-to-many relationship between **CREATOR** and **TREASURE_HUNT**, the foreign key is included in the **TREASURE_HUNT** relation referring to the primary key of **CREATOR**. The relationship features an attribute type, creation_date, which was added to the **TREASURE_HUNT** relation.
  - The 1-side of the [1..N] cardinality is not supported. There is no guarantee that every creator has at least one treasure hunt created, despite this being a requirement of the application.
- For the one-to-many relationship between **HUNTER** and **TEAM**, a foreign key team_ID referring to the primary key in **TEAM** was added to the **HUNTER** relation.
  - The relational model cannot enforce that a **TEAM** has at least two members, despite this being a requirement of the application.
- The many-to-many relationship between **HUNTER**, and **TREASURE_HUNT**, and **TEAM** and **TREASURE_HUNT** is a ternary relationship type. Thus, the relationship type, **HUNTS_IN**, was made into it's own relation. The foreign keys of **HUNTS_IN** refer to the primary keys of all the participating relations, and together make the primary key of **HUNTS_IN**. The **HUNTS_IN** relation also received the entry_fee attribute type of the relationship.
  - The relational model cannot enforce that at least three teams should participate in a treasure hunt. All other cardinalities are supported, however.

IMPLEMENTATION IN MYSQL

Table Creation

The following shows the MySQL code for the creation of the schema and tables.

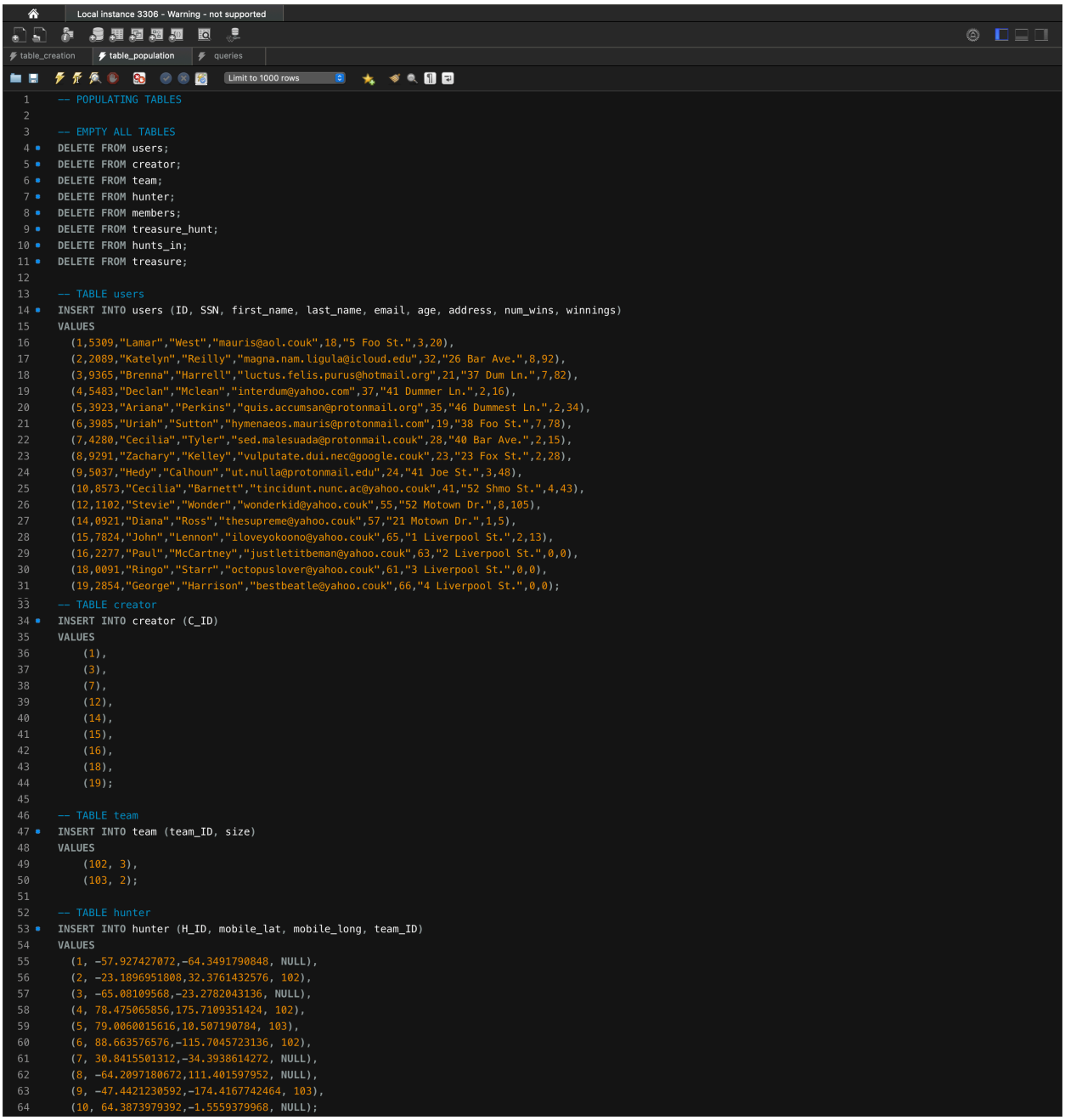



```sql
-- Create schema for database
CREATE SCHEMA IF NOT EXISTS treasure_hunt_app;

-- Use treasure_hunt_app schema
USE treasure_hunt_app;

-- DROP all tables before creation in case of updates
DROP TABLE IF EXISTS users, creator, team, hunter, members, treasure_hunt, hunts_in, treasure;

-- CREATE TABLES
-- Create users table
CREATE TABLE IF NOT EXISTS users (
    ID SMALLINT NOT NULL PRIMARY KEY,
    SSN CHAR(11),
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(50),
    age NUMERIC NOT NULL,
    address VARCHAR(255),
    num_wins SMALLINT,
    winnings SMALLINT
);

-- Create creator table
CREATE TABLE IF NOT EXISTS creator (
    C_ID SMALLINT NOT NULL PRIMARY KEY,
    FOREIGN KEY (C_ID) REFERENCES users (ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Create team table
CREATE TABLE IF NOT EXISTS team (
    team_ID SMALLINT NOT NULL PRIMARY KEY,
    size TINYINT
);

-- Create hunter table
CREATE TABLE IF NOT EXISTS hunter (
    H_ID SMALLINT NOT NULL PRIMARY KEY,
    mobile_lat FLOAT(12, 9) NOT NULL,
    mobile_long FLOAT(12, 9) NOT NULL,
    team_ID SMALLINT NULL,
    FOREIGN KEY (team_ID) REFERENCES team (team_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Create members table
CREATE TABLE IF NOT EXISTS members (
    team_ID SMALLINT NOT NULL,
    H_ID SMALLINT NOT NULL,
    FOREIGN KEY (team_ID) REFERENCES team (team_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (H_ID) REFERENCES hunter (H_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

```sql

-- Create treasure_hunt table
CREATE TABLE IF NOT EXISTS treasure_hunt (
    hunt_ID SMALLINT NOT NULL PRIMARY KEY,
    difficulty TINYINT,
    finder SMALLINT NULL,
    found_date DATE,
    end_date DATE,
    time_limit TIME,
    creation_date DATE,
    C_ID SMALLINT NOT NULL,
    FOREIGN KEY (finder) REFERENCES USERS (ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (C_ID) REFERENCES creator (C_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Create hunts_in table
CREATE TABLE IF NOT EXISTS hunts_in (
    H_ID SMALLINT NOT NULL,
    team_ID SMALLINT NULL,
    hunt_ID SMALLINT NOT NULL,
    PRIMARY KEY (H_ID, hunt_ID),
    entry_fee SMALLINT,
    FOREIGN KEY (H_ID) REFERENCES hunter (H_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (team_ID) REFERENCES team (team_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (hunt_ID) REFERENCES treasure_hunt (hunt_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Crate treasure table
CREATE TABLE IF NOT EXISTS treasure (
    hunt_ID SMALLINT NOT NULL PRIMARY KEY,
    prize_value SMALLINT,
    latitude FLOAT(12, 9) NOT NULL,
    longitude FLOAT(12, 9) NOT NULL,
    FOREIGN KEY (hunt_ID) REFERENCES treasure_hunt (hunt_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Populating Tables

The following shows the MySQL code for the population of tables. Some data was generated with aid from generatedata.com, while the rest was made manually.




```sql
-- POPULATING TABLES

-- EMPTY ALL TABLES
DELETE FROM users;
DELETE FROM creator;
DELETE FROM team;
DELETE FROM hunter;
DELETE FROM members;
DELETE FROM treasure_hunt;
DELETE FROM hunts_in;
DELETE FROM treasure;

-- TABLE users
INSERT INTO users (ID, SSN, first_name, last_name, email, age, address, num_wins, winnings)
VALUES
  (1,5309,"Lamar","West","mauris@aol.couk",18,"5 Foo St.",3,20),
  (2,2089,"Katelyn","Reilly","magna.nam.ligula@icloud.edu",32,"26 Bar Ave.",8,92),
  (3,9365,"Brenna","Harrell","luctus.felis.purus@hotmail.org",21,"37 Dum Ln.",7,82),
  (4,5483,"Declan","Mclean","interdum@yahoo.com",37,"41 Dummer Ln.",2,16),
  (5,3923,"Ariana","Perkins","quis.accumsan@protonmail.org",35,"46 Dummest Ln.",2,34),
  (6,3985,"Uriah","Sutton","hymenaeos.mauris@protonmail.com",19,"38 Foo St.",7,78),
  (7,4280,"Cecilia","Tyler","sed.malesuada@protonmail.couk",28,"40 Bar Ave.",2,15),
  (8,9291,"Zachary","Kelley","vulputate.dui.nec@google.couk",23,"23 Fox St.",2,28),
  (9,5037,"Hedy","Calhoun","ut.nulla@protonmail.edu",24,"41 Joe St.",3,48),
  (10,8573,"Cecilia","Barnett","tincidunt.nunc.ac@yahoo.couk",41,"52 Shmo St.",4,43),
  (12,1102,"Stevie","Wonder","wonderkid@yahoo.couk",55,"52 Motown Dr.",8,105),
  (14,0921,"Diana","Ross","thesupreme@yahoo.couk",57,"21 Motown Dr.",1,5),
  (15,7824,"John","Lennon","iloveyokoono@yahoo.couk",65,"1 Liverpool St.",2,13),
  (16,2277,"Paul","McCartney","justletitbeman@yahoo.couk",63,"2 Liverpool St.",0,0),
  (18,0091,"Ringo","Starr","octopuslover@yahoo.couk",61,"3 Liverpool St.",0,0),
  (19,2854,"George","Harrison","bestbeatle@yahoo.couk",66,"4 Liverpool St.",0,0);
-- TABLE creator
INSERT INTO creator (C_ID)
VALUES
    (1),
    (3),
    (7),
    (12),
    (14),
    (15),
    (16),
    (18),
    (19);

-- TABLE team
INSERT INTO team (team_ID, size)
VALUES
    (102, 3),
    (103, 2);

-- TABLE hunter
INSERT INTO hunter (H_ID, mobile_lat, mobile_long, team_ID)
VALUES
  (1, -57.927427072,-64.3491790848, NULL),
  (2, -23.1896951808,32.3761432576, 102),
  (3, -65.08109568,-23.2782043136, NULL),
  (4, 78.475065856,175.7109351424, 102),
  (5, 79.0060015616,10.507190784, 103),
  (6, 88.663576576,-115.7045723136, 102),
  (7, 30.8415501312,-34.3938614272, NULL),
  (8, -64.2097180672,111.401597952, NULL),
  (9, -47.4421230592,-174.4167742464, 103),
  (10, 64.3873979392,-1.5559379968, NULL);
```

```sql
-- TABLE members
INSERT INTO members (team_ID, H_ID)
VALUES
    (102, 2),
    (102, 4),
    (102, 6),
    (103, 5),
    (103, 9);

-- TABLE treasure_hunt
INSERT INTO treasure_hunt (hunt_ID, difficulty, finder, found_date, end_date, time_limit, creation_date, C_ID)
VALUES
    (201, 3, NULL, NULL, NULL,"13:32","03-08-23", 1),
    (203, 1, 8, "2023-12-31","2023-12-31","23:58","2023-11-27", 3),
    (207, 5, 10, "2023-07-26","2023-07-26","15:05","2023-06-09", 7),
    (202, 1, NULL, NULL, NULL,"15:05","2023-06-09", 12),
    (204, 2, 10, "2023-07-26","2022-01-11","15:05","2022-03-09", 14),
    (205, 3, NULL, NULL, NULL,"15:05","2023-06-09", 15),
    (206, 4, 10, "2023-01-26","2022-12-22","15:05","2022-06-15", 16),
    (208, 4, NULL, NULL, NULL,"15:05","2023-02-20", 18),
    (209, 5, NULL, NULL, NULL,"15:05","2022-03-27", 19);

-- TABLE hunts_in
INSERT INTO hunts_in (H_ID, team_ID, hunt_ID, entry_fee)
VALUES
    (2, 102, 201, 50),
    (4, 102, 201, 50),
    (6, 102, 201, 50),
    (5, 103, 201, 75),
    (9, 103, 201, 75);

-- TABLE treasure
INSERT INTO treasure (hunt_ID, prize_value, latitude, longitude)
VALUES
    (201, 125, 32.83455987512, -234.9757742764),
    (202, 110, 38.83451728512, 34.7751742764),
    (203, 70, 18.43916987512, 13.9527112764),
    (204, 85, 14.83455982512, 33.1786742764),
    (205, 20, 43.98745987512, -48.3098242764),
    (206, 35, 41.92455987512, 46.3190242764),
    (207, 140, 32.92409487512, 58.0988242764),
    (208, 30, 98.92944987512, 11.1828242764),
    (209, 25, 143.99745997512, -31.3023842764);

```

Queries

Here we give five example queries and their results.

```sql
-- QUERY 1 : View the users and the team they belong to.
SELECT u.ID, m.team_ID
FROM users AS u
INNER JOIN members AS m
ON (u.ID = m.H_ID);
```

query 1

ID	team_ID
2	102
4	102
6	102
5	103
9	103

```sql
-- Query 2 : Retrieve the users, the team the participate in, and the number of members on that team.
SELECT u.ID, m.team_ID, t.size
FROM team t
JOIN members m ON (t.team_ID = m.team_ID)
JOIN users u ON (m.H_ID = u.ID);
```

query 2

ID	team_ID	size
2	102	3
4	102	3
6	102	3
5	103	2
9	103	2

```sql
-- Query 3 : Which users have won at least 2 times?
SELECT ID, num_wins
FROM users
WHERE num_wins > 2;
```

query 3

ID	num_wins
1	3
2	8
3	7
6	7
9	3
10	4

```sql
-- Query 4 : Which users are in team 2?
SELECT ID
FROM users
WHERE ID IN (
    SELECT H_ID
    FROM members
    WHERE team_ID = 102);
```

query 4

ID
2
4
6

```sql
-- Query 5 : Retrieve all users participating in a hunt that they haven't found yet.
SELECT H_ID
FROM hunts_in h
JOIN treasure_hunt t ON (h.hunt_ID = t.hunt_ID)
WHERE t.found_date IS NULL;
```

query 5

H_ID
2
4
5
6
9

IMPLEMENTATION IN NOSQL

We implement the database in MongoDB. For the sake of simplicity, we only create the hunter, team, members, and treasure_hunt collection, including only necessary fields.

Collection Creation & Population

```
// Create collections
db.hunter.drop();
db.team.drop();
db.members.drop();
db.treasure_hunt.drop();

// Populate collections
db.hunter.insert({"_id":2, "team_ID":102, "winnings":40});
db.hunter.insert({"_id":4, "team_ID":102, "winnings":20});
db.hunter.insert({"_id":6, "team_ID":102, "winnings":5});
db.hunter.insert({"_id":5, "team_ID":103, "winnings":15});
db.hunter.insert({"_id":9, "team_ID":103, "winnings":0});

db.team.insert({"_id":102, "size":3});
db.team.insert({"_id":103, "size":2});

db.members.insert({"team_id":102, "H_ID":2});
db.members.insert({"team_id":102, "H_ID":4});
db.members.insert({"team_id":102, "H_ID":6});
db.members.insert({"team_id":103, "H_ID":5});
db.members.insert({"team_id":103, "H_ID":9});

db.treasure_hunt.insert({"_id":201, "finder":null, "C_ID":1});
db.treasure_hunt.insert({"_id":201, "finder":2, "C_ID":3});
db.treasure_hunt.insert({"_id":203, "finder":8, "C_ID":3});
db.treasure_hunt.insert({"_id":207, "finder":10, "C_ID":7});
```

Queries

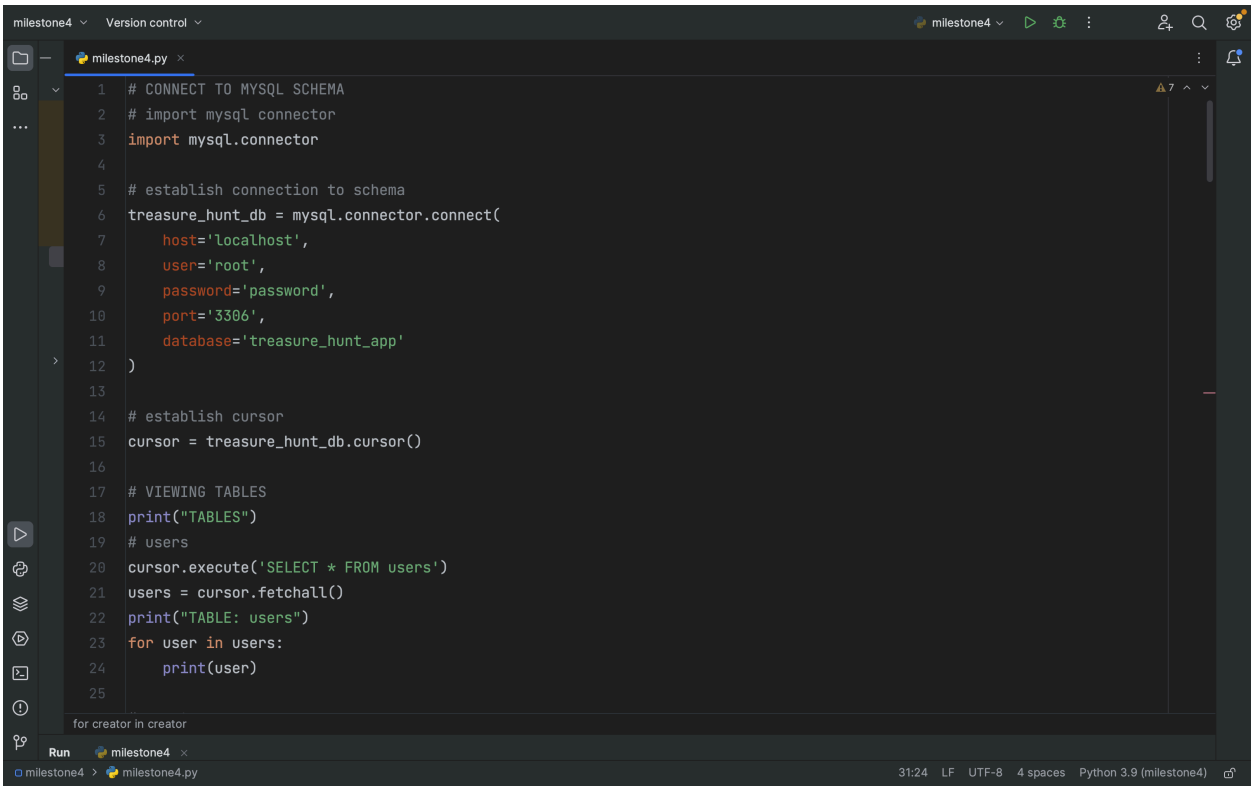
Here we run a few simple queries.

```
// Query 1 : Find all hunters who play on team 102
db.members.find({"team_id":102});

// Query 2 : Update hunter 2 to play on team 103
db.hunter.update({"_id":2}, {$set:{"team_ID":103}});

// Query 3 : Find hunts by creator 3 that have't been found
db.treasure_hunt.find({$and:[
   {"C_ID":3, "finder": {$exists: true, $eq: null}
}]});

// Query 4 : Find the total winnings of each team
db.hunter.aggregate([
   {$group:{_id:"$team_ID", total:{$sum:"$winnings"}}}
]);
```

```
// Query 1 Result
{"team_id" : 102, "H_ID" : 2}
{"team_id" : 102, "H_ID" : 4}
{"team_id" : 102, "H_ID" : 6}

// Query 3 Result
{"_id" : 203, "finder" : 8, "C_ID" : 3}

// Query 4 Result
// Notice that totals are affected by Query 2,
// which put hunter 2 on team 103
{"_id" : 102, "total" : 25}
{"_id" : 103, "total" : 55}
```

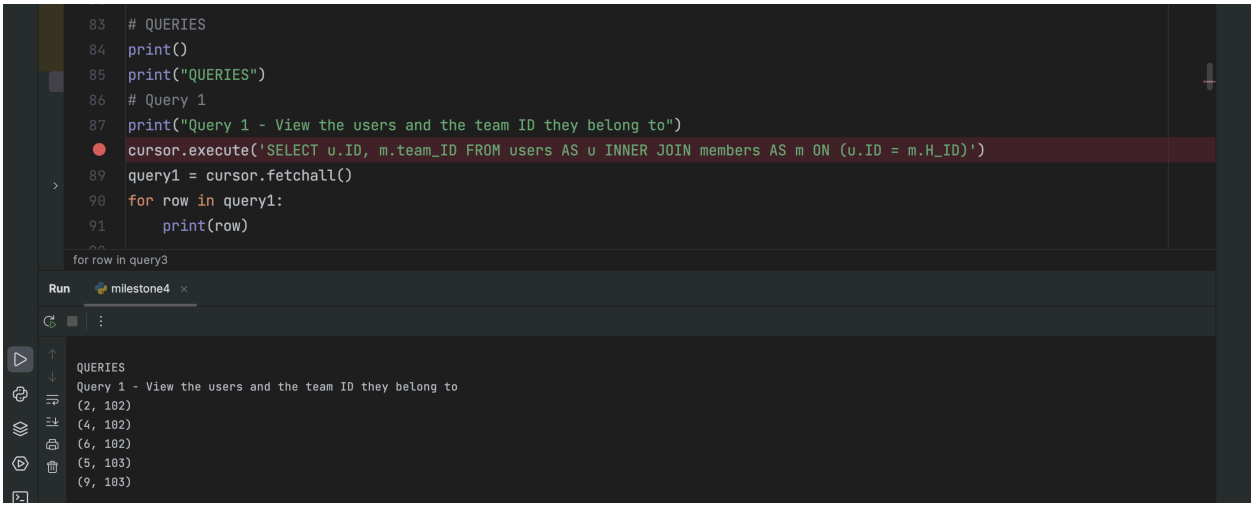
DATABASE ACCESS VIA PYTHON

The following python code was written to connect to the database schema.



```python
# CONNECT TO MYSQL SCHEMA
# import mysql connector
import mysql.connector

# establish connection to schema
treasure_hunt_db = mysql.connector.connect(
    host='localhost',
    user='root',
    password='password',
    port='3306',
    database='treasure_hunt_app'
)

# establish cursor
cursor = treasure_hunt_db.cursor()

# VIEWING TABLES
print("TABLES")
# users
cursor.execute('SELECT * FROM users')
users = cursor.fetchall()
print("TABLE: users")
for user in users:
    print(user)
```

The code for executing some query and it's result is shown below.

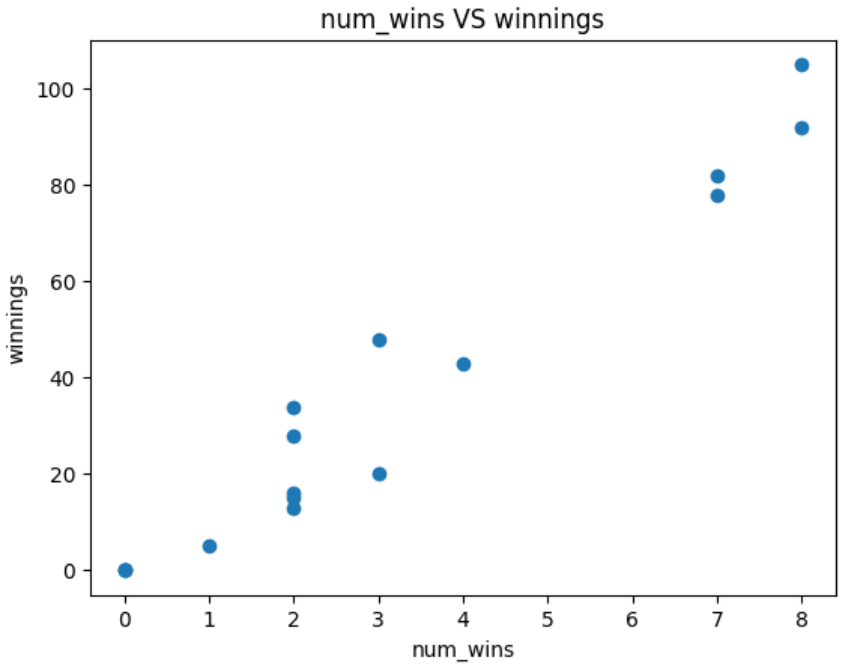


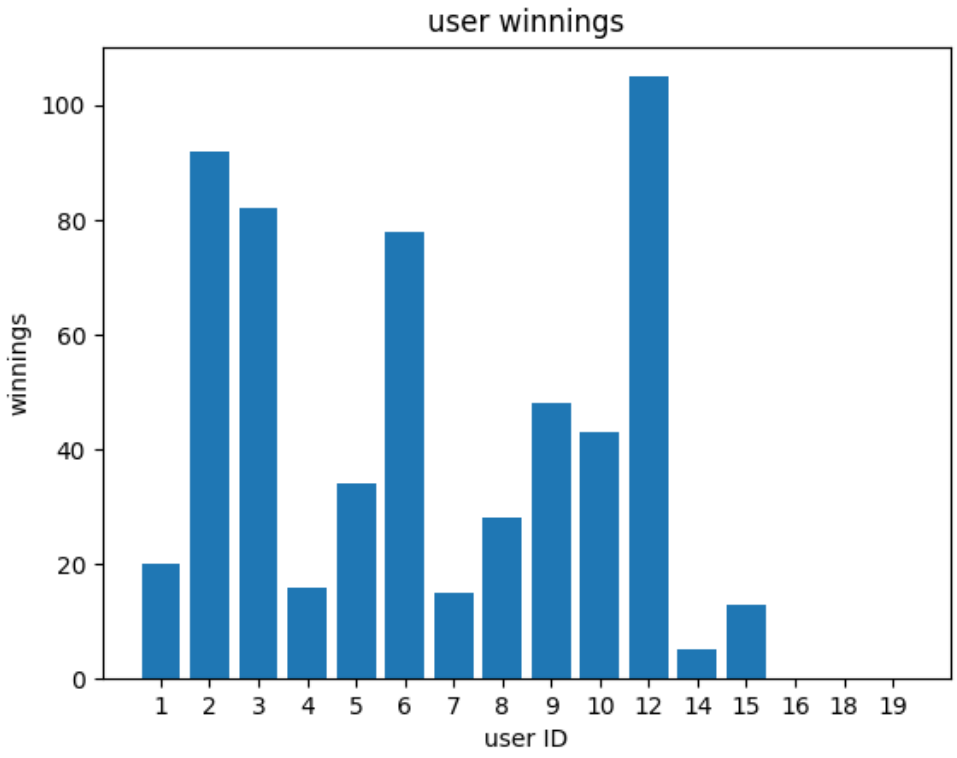
```python
# QUERIES
print()
print("QUERIES")
# Query 1
print("Query 1 - View the users and the team ID they belong to")
cursor.execute('SELECT u.ID, m.team_ID FROM users AS u INNER JOIN members AS m ON (u.ID = m.H_ID)')
query1 = cursor.fetchall()
for row in query1:
    print(row)
```

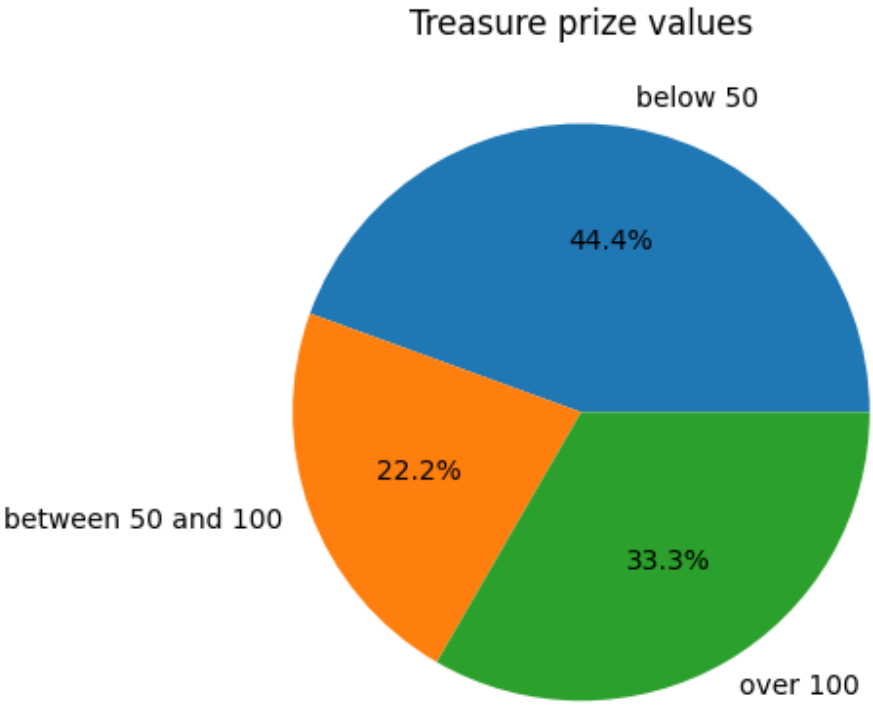
```
QUERIES
Query 1 - View the users and the team ID they belong to
(2, 102)
(4, 102)
(6, 102)
(5, 103)
(9, 103)
```

Plots

After converting the tables into pandas data frames, a few simple plots were made using matplotlib.






Treasure prize values
below 50
44.4%
22.2%
between 50 and 100
33.3%
over 100

CONCLUSION

This database provides a snapshot of what a public, commercial treasure hunting business application may utilize. We believe such an application would have great potential in the mobile application circuit, facilitating a communal outdoors hobby and operating almost completely by user interaction.

The database could be improved by expanding to include information on various geographical locations in which the application operates. This could facilitate the generation of hints during treasure hunts and aid in treasure location selection to ensure user safety.

The database does miss out on modeling several features inherent to the application. For example, although a time limit is incorporated into the treasure_hunt relation, further software code would be required to implement and ensure the feature. Moreover, the nature of user's identity as creators or hunters can become a tad dicey, as there is no assurance that users can switch between roles and cannot remain as both simultaneously. Again, this could be done with application code.