Assigned: 11 February 2021

Homework #2 – SQL and MongoDB (NoSQL)

EE 599: Spring 2021

Due: Thursday, 25 February 2021 at 23:59. Late penalty: 10% per 24-hours before 27 February at 23:59. Submission instructions will follow separately on canvas.

1. SQL league tracker

Create Python web application that uses a MySQL database to manage clashes (*i.e.* matches or games) between players in a small sports league. Your application should handle two entities: `Player` and `Clash`.

a. OVERVIEW

Clashes are contests between two players. There can be many simultaneous *active* clashes. An *active clash* is a clash that has not ended. Players may participate in any number of clashes but can be in only one active clash at a time. Players must pay the entry fee to participate in a clash. Players with insufficient balance to pay the entry fee cannot join a clash. Players may deposit funds to increase their balance.

The player with the most points when the clash ends is the winner. The winner receives the clash *prize*. Clashes cannot end if both players have the same number of points (*i.e.* no ties allowed). The clash ends immediately if one player is disqualified (DQ). The other player is the winner and receives the prize.

b. TECHNICAL

- Bind web server to port 3000.
- Create a remote (host:port) instance of mysql.
- Use `mysql.connector` python package. Other options may require additional dependencies. mysql-connector is maintained by the *official* MySQL Python team and is built with pure python.

c. SCHEMA AND DATATYPES

Assume the following base schema:

```
# USE db - note, db name specified in connection file

CREATE TABLE player (
  player_id     INT UNSIGNED NOT NULL AUTO_INCREMENT,
  fname         VARCHAR(20) NOT NULL CHECK (fname <> ''),
  lname         VARCHAR(20),
  is_active     BOOLEAN,
  balance_usd   DECIMAL(10,2) DEFAULT 0 CHECK (balance_usd >= 0),
  handed        ENUM('L', 'R', 'AMBI'),
  PRIMARY KEY (player_id)
);

CREATE TABLE clash (
  clash_id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
  player1_id    INT UNSIGNED NOT NULL,
  player2_id    INT UNSIGNED NOT NULL,
  attendance    INT UNSIGNED DEFAULT 0,
  entry_fee_usd DECIMAL(10,2) NOT NULL CHECK (entry_fee_usd >= 0),
  prize_usd     DECIMAL(10,2) NOT NULL CHECK (prize_usd >= 0),
  create_at     DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  end_at        DATETIME DEFAULT NULL,
  PRIMARY KEY (clash_id),
  CONSTRAINT pid1_fk FOREIGN KEY (player1_id) REFERENCES player(player_id) ON
DELETE CASCADE,
  CONSTRAINT pid2_fk FOREIGN KEY (player2_id) REFERENCES player(player_id) ON
DELETE CASCADE,
  CHECK (player1_id <> player2_id)
);

CREATE TABLE clash_point (
  player_id    INT UNSIGNED NOT NULL,
  clash_id     INT UNSIGNED,
  points       INT UNSIGNED,
  is_dq        BOOLEAN,
  event_at     DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (clash_id, event_at),
  CONSTRAINT cid_fk FOREIGN KEY (clash_id) REFERENCES clash(clash_id) ON DELETE
CASCADE,
  CONSTRAINT pid_fk FOREIGN KEY (player_id) REFERENCES player(player_id) ON
DELETE CASCADE
);
```

You may extend the schema but DO NOT MODIFY ANY EXISTING ATTRIBUTES (see: `GET /schema/extend`). The autograder first calls the `/admin/pre` endpoint so you can run SQL "setup" statements prior to starting tests. Real (*i.e.* the human) graders will also review your schema for violations and other implementation issues.

Currency: Represent all currencies as USD (*i.e.* $xxx.xx). Use DECIMAL numbers with exactly two precision digits (*i.e.* dollars and cents). Numbers or strings with more than two digits are invalid. Append zeros as necessary if less than two digits, e.g. 1.332 is invalid (too many digits) but 1.33, 1.3, 1.0, and 1 are all valid (*i.e.* append zeros).

Time/Date: Report all times and dates as UTC ISO-8601 (always both date and time).

Boolean: Accept (case insensitive) "1", "true", and "t" as **true** Boolean values. Assume all other non-empty or null values are **false**.

Player Name: Use "fname lname" as player name. If lname is empty or null use only "fname". The name must not include any trailing spaces.

d. API SPECIFICATION

Your application must implement the following API (see ***Response Syntax*** for entity detail). Assume the first response code if the response meets multiple conditions.

- **`GET /player`**

  Return: Array of all **active** `Players`. Sort by player name ASC (i.e. "A to Z").

  Response code: `200`

- **`GET /player/[pid]`**

  Return: `Player[pid]`.

  Response code: `200` if exist, `404` if not exist.

- **`POST /player?fname=&lname=&handed=[enum]&initial_balance_usd=[currency]`**

  Add a new active player. The query string may contain none, some, or all the parameters. First and last names may contain only letters. Handed should be one of (case-insensitive): "left", "right", or "ambi". INSERT the player only if it satisfies the schema. Else fail.

  Response code:

  `303` redirect on success to `GET /player/[pid]`.

  `422` error on failure: body must be string that includes all invalid field names, e.g. "invalid fields: initial_balance_usd" or "invalid fields: handed, fname, initial_balance_usd", etc. You do not need to report the reason.

- **`POST /player/[pid]?active=[bool]&lname=`**

  Update `Player[pid]`. The query string may contain none, some, or all the parameters. UPDATE the player only if it satisfies the schema.

  Response code:

  `303` redirect on success: `GET /player/[id]`.

  `200` if exist, `404` if not exist.

- **`POST /deposit/player/[pid]?amount_usd=[currency]`**

  Add **positive** currency to `Player[pid]` balance.

  Return: `PlayerBalance[pid]`.

  Response code:

`200` on success, `404` if player does not exist, and `400` if invalid amount.

✓ - **`GET /clash`**

  Return: Array of `Clashes`. Return all active clashes sort by prize_usd DESC (i.e. "largest first") and then the four most recently ended inactive clashes sorted by end_at DESC ("newest first").

  Response code: `200`

- **`GET /clash/[cid]`**

  Return: `Clash[cid]`.

  Response code: `200` if exist, `404` if not exist.

- **`POST /clash?pid1=&pid2=&entry_fee_usd=[currency]&prize_usd=[currency]`**

  Start a new Clash. Pid1 and Pid2 must exist, have balance sufficient to cover the entry fee, and not be in an active clash already.

  Response code:

  `303` redirect on success to `GET /clash/[cid]`.

  `404` if player1 or player2 does not exist.

  `409` if player1 or player2 already in an active clash.

  `402` if insufficient account balance (either player).

  `400` else.

- **`POST /clash/[cid]/award/[pid]?points=`**

  Points must be positive integer. Player must be in the clash and clash must be active.

  Return: `Clash[cid]`.

  Response code:

  `200` on success, `404` if player or clash does not exist, `409` if clash not active, `400` else.

- **`POST /clash/[cid]/end`**

  End an active clash. Clash must exist and be active. One player points must be higher than the other player points.

Return: `Clash[cid]`.

Response Code:

`200` if success, `404` if not exist, `409` if clash not active or points tied.

- **`POST /clash/[cid]/disqualify/[pid]`**

  Disqualify a player from clash and end the clash. Clash must be active and player must be in the clash.

  Return: `Clash[cid]`.

  Response code:

  `200` on success, `404` if player or clash does not exist, `409` if clash not active, `400` else.

- **`GET /ping`**

  Response code: `204` (always).

- **`POST /admin/pre`**

  Apply schema additions. The autograder creates the database and base schema before it runs your script. It will then issue `GET /admin/pre`. You may use the call to trigger any schema create/alter statements.

  Return: string “`OK`” (no quotes).

  Response code: `200` (always).

e. RESPONSE SYNTAX

Use JSON for all (non-empty) responses. Use the following syntax for entity response.

`Player[pid]`

```
{
  pid:            int       player id
  name:           string    “fname lname” - no trailing spaces
  handed:         string    left|right|ambi
  is_active       boolean
  num_join:       int            number of clashes
  num_won:        int            number of clashes won
  num_dq          int       number of disqualifications
  balance_usd     string    currency string
  total_points    int       total number of points in all clashes
  rank            int       rank num_won, 1 = highest to N = lowest
                            (assume no tie)
  spec_count      int       total number of spectators to see player clashes
  total_prize_usd int            total prize for player (currency string)
  efficiency      float     % of *completed* clashes won
  in_active_clash int       cid of active clash, else null
}
```

`Clash[cid]`

```
{
  cid:          int
  p1_id:        int
  p1_name:      string      (see Player.name)
  p1_points:    int
  p2_id:        int
  p2_name:      string      (see Player.name)
  p2_points:    int
  winner_pid:   int|null    null if active
  is_active:    boolean
  prize_usd:    string
  age:          int         seconds since create
  ends_at:      string      ISO-8601 (date+time)
  attendance    int
}
```

`PlayerBalance[pid]`

```
{
  old_balance_usd: string       (currency string)
  new_balance_usd: string       (currency string)
}
```

f. DATABASE CONNECTION

Read the database connection info from a JSON file. Use the JSON file path `./config/mysql.json` relative to your script. Exit with code 2 if the file is invalid: empty, does not exist, invalid JSON, etc. Exit with code 4 if the file is valid but the MySQL database connection fails for any reason.

```
{
  host: "…",
  port: "…",
  user: "…",
  pass: "…",
  db:   "…"
}
```

2. **NoSQL (Mongo) request log**

Use MongoDB to create an audit log for every incoming request.

Use a UUID (v4) to identify incoming request. Add a new document to the "`request_log`" collection for each request and include at least the following fields:

```
{
  request_id:           mongo UUID()
  is_sensitive:         true
  request_time:         mongo Date()
  request_duration_ms:  int
  request_ip:           string
  request_user_agent:   string
  request_method:       string
  request_headers:      {Header:Value,…}
  request_body:         string
  request_path:         string                (no query string)
  request_query_string  string
  host:                 string                (server hostname)
  response_headers:     {Header:Value}
  response_http_code:   int
  response_body:        string
  response_is_error:    boolean
}
```

Then add the following to every <u>JSON</u> response:

```
{
  […]         (regular of response),
  req: {
    server_time:  UTC ISO-8601                (date and time)
    request_uuid:                             (hex string)
  }
}
```

DATABASE CONNECTION

Read the database connection info from a JSON file. Use the JSON file path `./config/mysql.json` relative to your script. Exit with code 2 if the file is invalid: empty, does not exist, invalid JSON, etc. Exit with code 5 if the file is valid but MongoDB connection fails for any reason.

```
{
  host: "…",
  port: "…",
  user: "…",
  pass: "…",
  db:   "…"
}
```