Ekin	Daniele	Edoardo
Oral	Venere	Cecca
29421	36745	36779

Judo Association Application: Phase 3

Introduction

Our project is an application for managing judo associations. Some requirements for our application are that it stores data about judokas, dojos, judo associations, and events these associations hold. In this phase, we have created a web application demonstrating our triggers, stored procedures, and including a support ticket system.

Triggers

`Check_date`, `Check_date_upd`

```
DELIMITER //
CREATE TRIGGER Check_date BEFORE INSERT ON JudoMatch
FOR EACH ROW
BEGIN
  DECLARE start_event DATE;
  DECLARE end_event DATE;

  SELECT startDate INTO start_event FROM JudoEvents WHERE eId = NEW.eId LIMIT 1;
  SELECT endDate INTO end_event FROM JudoEvents WHERE eId = NEW.eId LIMIT 1;

  IF (NEW.mDate NOT BETWEEN start_event AND end_event) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Match date is incorrect. Please ensure the date is within the event
range.';
  END IF;
END//

CREATE TRIGGER Check_date_upd BEFORE UPDATE ON JudoMatch
FOR EACH ROW
BEGIN
  DECLARE start_event DATE;
  DECLARE end_event DATE;

  SELECT startDate INTO start_event FROM JudoEvents WHERE eId = NEW.eId LIMIT 1;
  SELECT endDate INTO end_event FROM JudoEvents WHERE eId = NEW.eId LIMIT 1;

  IF (NEW.mDate NOT BETWEEN start_event AND end_event) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Match date is incorrect. Please ensure the date is within the event
```

```
range.';
  END IF;
END//
DELIMITER ;
```

Code for Check_date and Check_date_upd triggers.

Cases

1. “Outside range query” executes a query inserting a JudoMatch instance with the date outside of the duration of the referenced Event. The expected result is an error with no changes to the database state.
2. “Inside range query” executes a query with a valid date. The expected result is that the query is successful and the JudoMatch instance is added to the database.

check_teach, check_teach_upd

```
DELIMITER //

CREATE TRIGGER check_teach BEFORE INSERT ON TeachIn
FOR EACH ROW
BEGIN
  DECLARE belt_check ENUM("white","yellow","orange","green", "blue","brown", "black");
  SELECT J.belt into belt_check FROM Judoka J where J.jId = NEW.jId;

  IF  belt_check <> "black" THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "This Judoka cannot be inert into the
teachers, since they don't have a black belt";
  END IF;
END//


CREATE TRIGGER check_teach_upd BEFORE UPDATE ON TeachIn
FOR EACH ROW
BEGIN
  DECLARE belt_check ENUM("white","yellow","orange","green", "blue","brown", "black");
  SELECT J.belt into belt_check FROM Judoka J where J.jId = NEW.jId;

  IF  belt_check <> "black" THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "This Judoka cannot be inert into the
teachers, since they don't have a black belt";
  END IF;
END//
DELIMITER ;
```

Code for check_teach and check_teach_upd triggers.

Cases

1. “Invalid teacher” executes a query inserting a TeachIn relationship from a non black belt having judoka to a dojo. The expected result is an error.
2. “Valid teacher” executes a query inserting a TeachIn relationship from a black belt judoka to a dojo. The expected result is that the query is successful and the TeachesIn instance is added to the database.

`check_correctness`, `check_correctness_upd`

```
DELIMITER //
CREATE TRIGGER check_correctness BEFORE INSERT ON PlayedScore
FOR EACH ROW
BEGIN
  IF NEW.kScore IS NOT NULL  AND (NEW.ippon IS NOT NULL OR NEW.wazari IS NOT NULL OR NEW.yuko
IS NOT NULL) THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "ERROR: Score format is incorrect try
checking if the score is a kata or 1v1 and if the attributes are according";
  END IF;

  IF NEW.kScore IS NULL  AND NEW.ippon IS NULL AND NEW.wazari IS NULL AND NEW.yuko IS NULL
THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "ERROR: Score format is incorrect try
checking if the score is a kata or 1v1 and if the attributes are according";
  END IF;
END//

CREATE TRIGGER check_correctness_upd BEFORE UPDATE ON PlayedScore
FOR EACH ROW
BEGIN
  IF NEW.kScore IS NOT NULL  AND (NEW.ippon IS NOT NULL OR NEW.wazari IS NOT NULL OR NEW.yuko
IS NOT NULL) THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "ERROR: Score format is incorrect try
checking if the score is a kata or 1v1 and if the attributes are according";
  END IF;
  IF NEW.kScore IS NULL  AND NEW.ippon IS NULL AND NEW.wazari IS NULL AND NEW.yuko IS NULL
THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "ERROR: Score format is incorrect try
checking if the score is a kata or 1v1 and if the attributes are according";
  END IF;
END //
DELIMITER ;
```

Code for check_correctness and check_correctness_upd triggers.

Cases

1. "Invalid: No score attributes" executes a query inserting a invalid instance of PlayedScore

where none of the score attributes are filled. The expected result is an error, and the instance not in the database.
2. "Invalid: Attributes of both" executes a query inserting a invalid instance of PlayedScore where attributes for both score types are present. The expected result is an error, and the instance not in the database.
3. "Valid: 1v1 Score" executes a query inserting a valid instance of PlayedScore where all 1v1 score attributes are null. The expected result is that the instance is added to the database.
4. "Valid: Kata score" executes a query inserting a valid instance of PlayedScore where the kata score attribute is null. The expected result is that the instance is added to the database.

Stored Procedures

`yearly_subs`

```
DELIMITER //
CREATE PROCEDURE yearly_subs(IN Start_date DATE, IN End_date DATE)
BEGIN
   SELECT *
   FROM Judoka
   WHERE startDate BETWEEN Start_date AND End_date ;
END //
DELIMITER ;
```

Code for yearly_subs procedure.

Inputs

1. `Start_date`: Year input box in the web page
2. `Start_date`: Year input box in the web page

`check_winner`

```
DELIMITER //
CREATE PROCEDURE check_winner(IN curr_id INT)
BEGIN
   SELECT PS.mId, ippon*10 + wazari*5 + yuko as Points, ippon, wazari, yuko, J.jName
   FROM PlayedScore PS, JudoMatch JM, Judoka J
   WHERE PS.mId = curr_id AND PS.forfeit <> curr_id AND PS.mId = JM.mId AND J.jId = PS.jId
   ORDER BY Points DESC LIMIT 1;
END //

CREATE PROCEDURE versus()
BEGIN
   SELECT P1.mId, J1.jName as player1, J2.jName as player2
   FROM PlayedScore P1, PlayedScore P2, Judoka J1, Judoka J2
   WHERE J1.jId < J2.jId AND P1.jId < P2.jId AND P1.mId = P2.mId AND P1.jId = J1.jId AND
P2.jId = J2.jId AND P1.kScore IS NULL;
END//
```

```
DELIMITER;
```

Code for check_winner and versus procedures.

Inputs

1. `curr_id`: “Match id” dropdown on web page. The options are populated using the `versus` procedure.

`maxProfitEvent`

```
DELIMITER //
CREATE PROCEDURE maxProfitEvent()
BEGIN
   SELECT R.eId, COUNT(R.rStatus) * JE.price as gain, JE.eName
   FROM Request R, JudoEvents JE
   WHERE R.rStatus = 'approved' AND JE.eId = R.eId
   GROUP BY R.eId, JE.eName
   ORDER BY gain DESC
   LIMIT 1;
END //
DELIMITER ;
```

Code for maxProfitEvent procedure.

Inputs

No inputs.

MongoDB Queries

`Find Distinct Usernames of Active Tickets`

```
$distinct = new MongoDB\Driver\Command([
  'distinct' => 'tickets',
  'key' => 'username',
  'query' => [
      'status' => true,
  ]
]);
$usernames = $mongo->executeReadCommand(getenv("MONGO_DATABASE"), $distinct)
  ->toArray()[0]
  ->values;
```

Get Tickets from User

```
$results = $mongo->executeQuery(getenv("MONGO_DATABASE") . '.tickets', new
MongoDB\Driver\Query([
  'username' => $username,
  'status' => true,
]));
```

Get Ticket with ID

```
$tickets = $mongo->executeQuery(getenv("MONGO_DATABASE") . '.tickets', new
MongoDB\Driver\Query(['_id' => new \MongoDB\BSON\ObjectId($id)]));
```

Create New Ticket

```
$bulk = new MongoDB\Driver\BulkWrite();
$bulk->insert([
  'username'=> $username,
  'body'=> $body,
  'created_at' => date("c"),
  'status'=> TRUE,
  'comments'=> [],
]);
$result = $manager->executeBulkWrite(getenv("MONGO_DATABASE") . ".tickets", $bulk);
```

Add Comment to Ticket

```
$bulk = new MongoDB\Driver\BulkWrite();
$bulk->update(['_id' => new \MongoDB\BSON\ObjectId($_GET['id'])], [
 '$push' => [
   'comments' => [
     'username' => $username,
     'comment' => $comment_body,
     'created_at' => date("c"),
   ]
 ]
]);
$r = $mongo->executeBulkWrite(getenv("MONGO_DATABASE") . ".tickets", $bulk);
```

Get Active Tickets

```
$tickets = $mongo->executeQuery(getenv("MONGO_DATABASE") . '.tickets', new
MongoDB\Driver\Query(['status' => true]));
```

Deactivate Ticket with ID

```
$bulk = new MongoDB\Driver\BulkWrite();
$bulk->update(['_id' => new \MongoDB\BSON\ObjectId($id)], [
  '$set' => [
    'status' => false,
  ]
]);
$r = $mongo->executeBulkWrite(getenv(name: "MONGO_DATABASE") . ".tickets", $bulk);
```