Ekin Oral 29421
Daniele Venere 36745
Edoardo Cecca 36779

# Judo Association Application: Phase 2

## 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 implemented triggers to enforce consistency based on our requirements and stored procedures for some often needed operations.

## Triggers

We implemented three pairs of insert and update triggers for ensuring consistency.

### `Check_date`, `Check_date_upd`

When judo matches are inserted into `JudoMatch`, they must be associated with a judo event which has specified start and end dates. Based on our real-world requirements, a constraint is that a match date must fall inside the duration of the event it is associated with. Before a row is inserted into or updated in `JudoMatch`, these two triggers verify that the date is valid given the constraint, and reject the insert or update if it is invalid, ensuring consistency.

### `check_teach`, `check_teach_upd`

The Judoka-Dojo relationship where the judoka is an instructor at the dojo is represented by the `TeachIn` relation. According to our requirements, a judoka must be at a black belt level in order to be allowed to teach at a dojo. Before a row is inserted or updated, these two triggers ensure that the referenced judoka has a black belt by rejecting the operation if not.

### `check_correctness`, `check_correctness_upd`

According to our ER model, each judoka in a match has one match score entity related to them. A match score can either be a 1v1 score or a kata score, represented by an is-a relationship in our ER model. We chose to translate the match score entity as a single relation `PlayedScore` having attributes representing both 1v1 and kata score, and requiring that one and only one of the types of attributes may be set on an instance. Before a row is inserted into `PlayedScore`, these triggers will ensure that this constraint is satisfied and reject the operation if not.

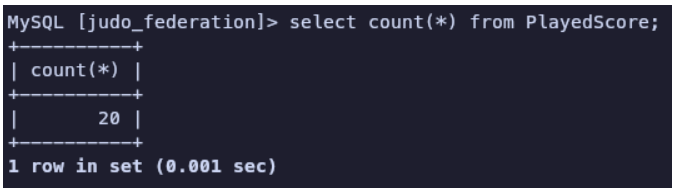
Included below is the script for trigger `check_correctness`, which runs when an insert is performed on `PlayedScore`. For each new row, if the row has non-NULL values for both 1v1 score attributes and kata score attribute, the insertion is rejected with an error message. The insertion will also be rejected in the same way if the attributes for both types of score are NULL. An

example of the trigger working is also below. In the example execution, the insertion is rejected due to kScore not being NULL and 1v1 score attributes also not being null.

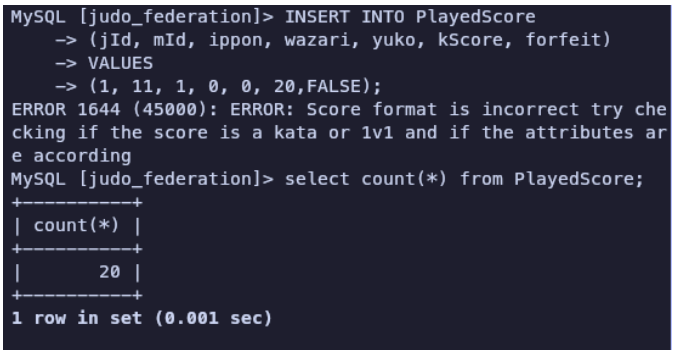
```
  DELIMITER //
   CREATE TRIGGER check_correctness BEFORE INSERT ON PlayedScore
   FOR EACH ROW
   BEGIN
    IF NEW.kScore IS NOT NULL  AND (NEW.ippon IS NOT NULL OR NEW.wazari
IS NOT NULL OR NEW.yuko IS NOT NULL) THEN
     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "ERROR: Score format is
incorrect 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//
```

excerpt from `scripts/triggers/trigger_3/trigger_3.sql`: trigger `check_correctness`

Before insert operation




After insert operation




# Stored Procedures

## `yearly_subs`

This stored procedure takes in two date parameters and returns the judokas that started between these dates.

### check_winner

This stored procedure takes in one integer parameter, a match id, and returns the match id, total points, point composition and the name of the judoka who won the match. This stored procedure implements the calculation to get total score from score components, as well as checking for forfeits. The script for this is included below.

```
  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 //
```

excerpt from `scripts/stored_procedures/stored_procedure2.sql`: stored procedure `check_winner`

### maxProfitEvent

This stored procedure returns the id, gross income, and name of judo event in the database that made the most gross income from participant entry fees. The gross income is found by multiplying the fee for the event with the number of approved requests to participate in the event.