YACM – YET ANOTHER CYCLING MANAGER

AUTHORS AND SUBJECT

This document describes the YACM, a system that allows to manage cycling events, their stages, participants, teams, results, equipment, and others.

PROJECT NAME	CLASS - GROUP
YACM – Yet Another Cycling Manager	P2 - G10

NAME	N. MEC	PERCENTAGE OF WORK
Paulo Vasconcelos	84897	50%
Pedro Teixeira	84715	50%

VERSION

Version 1 | June 06th 2019

CONTENTS

README

The submitted program already connects to the class server. If changes are needed (such as changing the username and password used to connect to the server), change the UserID and Password on file *DatabaseHandler.cs*, method *GetSQLServerConnection,* case "Class Server".

The program is prepared for, during execution, change the SQL Server to be connected to. It is possible to choose between the class server, the authors' personal servers and a hard coded server ("Other" option), which can be changed on line 54 of file *DatabaseHandler.cs*.

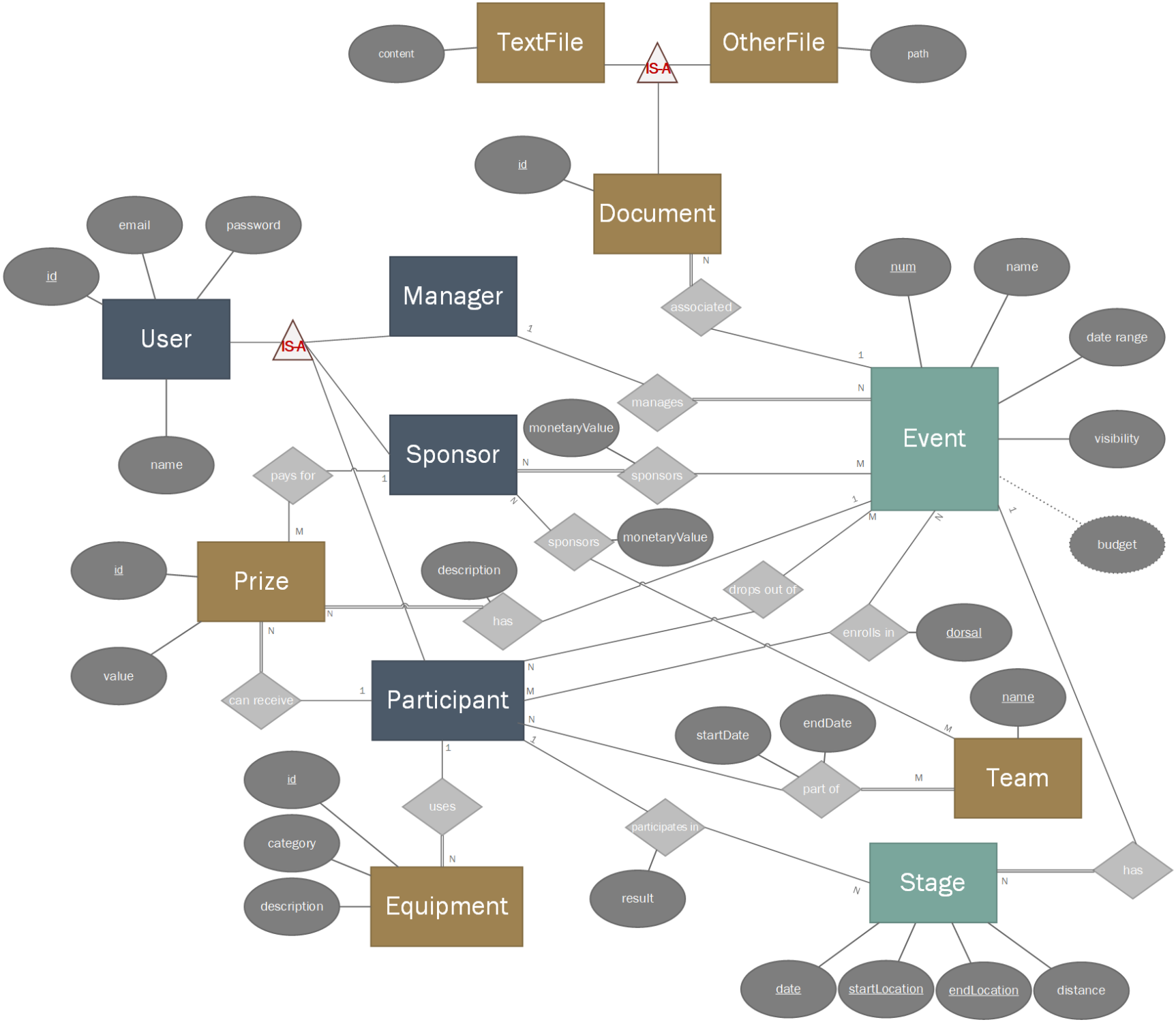
REQUIREMENTS ANALYSIS

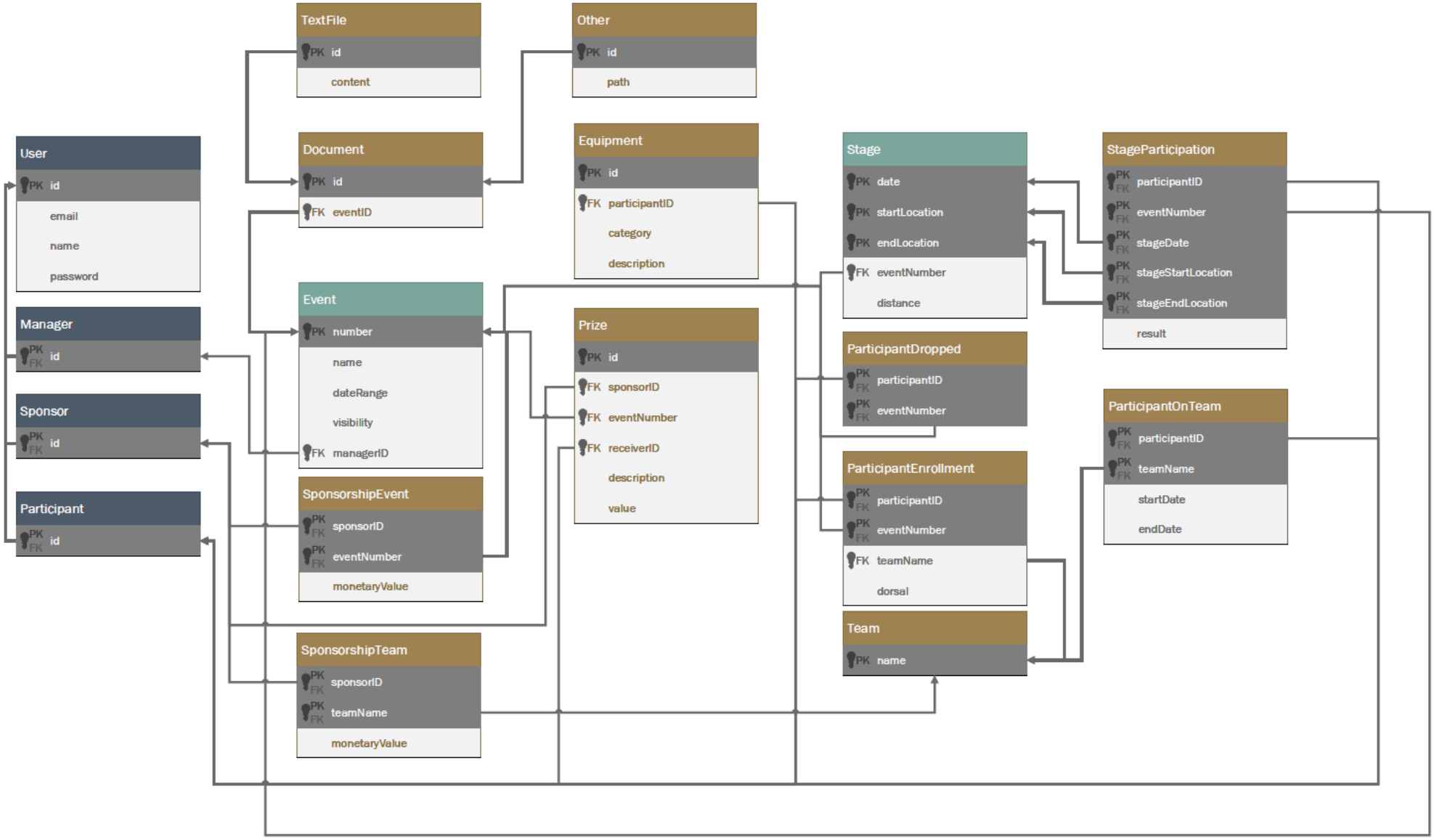
After interviewing a student in Sports Area and a team coach:

- A **problem** was found: **The organization of a simple informal sports event entails the tedious, time consuming and error-prone management of many informations** (registrations, teams and tickets, the creation of requests for authorizations to the competent authorities, the creation of a set of supporting documents for all these tasks and the dissemination of the same event).
- Which offers an **opportunity**: Why not **decrease the amount of work associated with the organization of these sports events** and consequently decrease the time, the number of people used and possible human errors.

The system should allow the users the following features:

User	Feature
Everyone	• Create an account either as Manager, Participant, Sponsor or Observer.
Manager	• Create and modify an event and its basic information > name, date, visibility, stages • Create documents associated with an event > such as authorizations, etc. • Manage equipment • Invite an observer to an event
Participant	• Enroll in an event • Being part of a team • Perform in a stage • Win a prize • Use equipment
Sponsor	• Sponsor an event • Sponsor a prize
Observer	• See Event and Stages Info > date, local


content
TextFile
IS-A
OtherFile
path
id
Document
email
password
id
User
IS-A
Manager
num
name
associated
date range
manages
Event
visibility
monetaryValue
Sponsor
sponsors
name
pays for
budget
sponsors
monetaryValue
id
Prize
description
has
drops out of
enrolls in
dorsal
value
can receive
Participant
name
startDate
endDate
Team
part of
id
uses
participates in
has
category
Equipment
Stage
description
result
date
startLocation
endLocation
distance


TextFile
PK id
content
Other
PK id
path
User
PK id
email
name
password
Document
PK id
FK eventID
Equipment
PK id
FK participantID
category
description
Stage
PK date
PK startLocation
PK endLocation
FK eventNumber
distance
StageParticipation
PK FK participantID
PK FK eventNumber
PK FK stageDate
PK FK stageStartLocation
PK FK stageEndLocation
result
Manager
PK FK id
Event
PK number
name
dateRange
visibility
FK managerID
Prize
PK id
FK sponsorID
FK eventNumber
FK receiverID
description
value
ParticipantDropped
PK FK participantID
PK FK eventNumber
Sponsor
PK FK id
ParticipantOnTeam
PK FK participantID
PK FK teamName
startDate
endDate
Participant
PK FK id
SponsorshipEvent
PK FK sponsorID
PK FK eventNumber
monetaryValue
ParticipantEnrollment
PK FK participantID
PK FK eventNumber
FK teamName
dorsal
Team
PK name
SponsorshipTeam
PK FK sponsorID
PK FK teamName
monetaryValue

SQL DDL

See SQL Scripts\SQL_DDL.sql.

SQL DML

The interactions between the system that weren't complex enough to require a Stored Procedure or an UDF were written as simple SQL instructions directly in the C# code. Therefore, there is no file to be analyzed.

INDEXES

See SQL Scripts\Indexes.sql for the SQL code.
In this system there wasn't the need for many indexes since each Entity Primary Keys would, most of the time, are used to solve the problem.
However, we created indexes for the following Entities, applied to attributes commonly used on selection queries:

- Event → managerID (since events to be presented on the UI are selected using the manager ID)
- Prize → eventNumber (since prizes are presented per event)
- Stage → eventNumber (since stages are selected per event)
- User → email (since users are searched using the email provided in the login UI)

TRIGGERS

See SQL Scripts\Triggers.sql.
When creating instances of an Entity, we opted for using Stored Procedures. We used Triggers when deleting an object on an Entity derived from another. The cases where this happened were:

- Deleting Documents (TextFile or OtherFile);
- Deleting User (Manager, Sponsor or Participant).

STORED PROCEDURES

See SQL Scripts\StoredProcedures.sql.
We used Stored Procedures when creating an object on an Entity derived from another. The cases where this happened were:

- Creating Documents (TextFile or OtherFile);
- Creating User (Manager, Sponsor or Participant).

USER DEFINED FUNCTIONS

See SQL Scripts\UDFs.sql.
We used UDFs when retrieving some more complex information from a variety of tables. This often required JOINs and UNIONs. The cases of use were the following:

- Retrieving the events managed by the user;
- Retrieving the events not managed by the user (in this case, we have one UDF that retrieves all other events and another that only retrieves those that are publicly available);
- Retrieving event-related information:
    - Prizes;
    - Teams status;
    - Event Documents.

DATASET

For testing purposes, we created a Dataset using Microsoft Excel (Dataset folder). Since some of the dataset is randomly generated using RAND or RANDBETWEEN, a static version was created (YACM-Dataset)

VIDEO

Available at https://uapt33090-my.sharepoint.com/:v:/g/personal/pedro_teix_ua_pt/EV77AT8nTdhPq-kTMm2cLsUBU5I2JnE4g-m7iF6AhMzhYw?e=hIxoF4

CONCLUSIONS

Key aspects we retained from this work:

- We retained the whole process for creating an Application with a proper Database behind – from the Requirements Analysis to the creation of the conceptual model and the relational model to the creation of an abstraction layer and finalizing in the execution of SQL Queries/UDFs/SPs on the application
- The relational model is adequate to the context of our problem – the keywords said during the interviews were a perfect match to the main entities in the ERD.
- Some relationships between entities would have to be reconsidered, especially on the Users – the concept of Manager, Participant and Equipment should be per event (meaning the creation of relation tables like ManagerEvent associating event IDs with user IDs), not per user (like it is implemented).
- How to store files and their info would also need a review – store the information as a blob? (probably not a good idea ☹); keep the current solution? other solutions we couldn't think of?