Peter Maynard

Llandwp Sports Club
CS27020 Assignment

11th March 2011

Table of Contents

Introduction

This report is to show the correct way to implement enterprise rules for a database. This report will cover the use of primary and foreign keys, as well as other database and table constraints that help with consistency through out the database.

This report will use design tools such as Unified Modeling Language, UML, to help represent the problem and results to the reader. The report is broken into four parts, these are Original implementation. This explains why the original implementation failed. The Correct Implementation, which explains the use of primary and foreign keys that where used to create the final correct result. The third section will cover what SQL commands where used to create the implementation. The final section will contain the tests and results of the SQL.

Original Implementation

See illustration below, which is a UML Class Diagram that illustrates the relational model implemented by the SQL CREATE TABLE commands that was provided with the assignment specification. See Appendix A for the SQL.

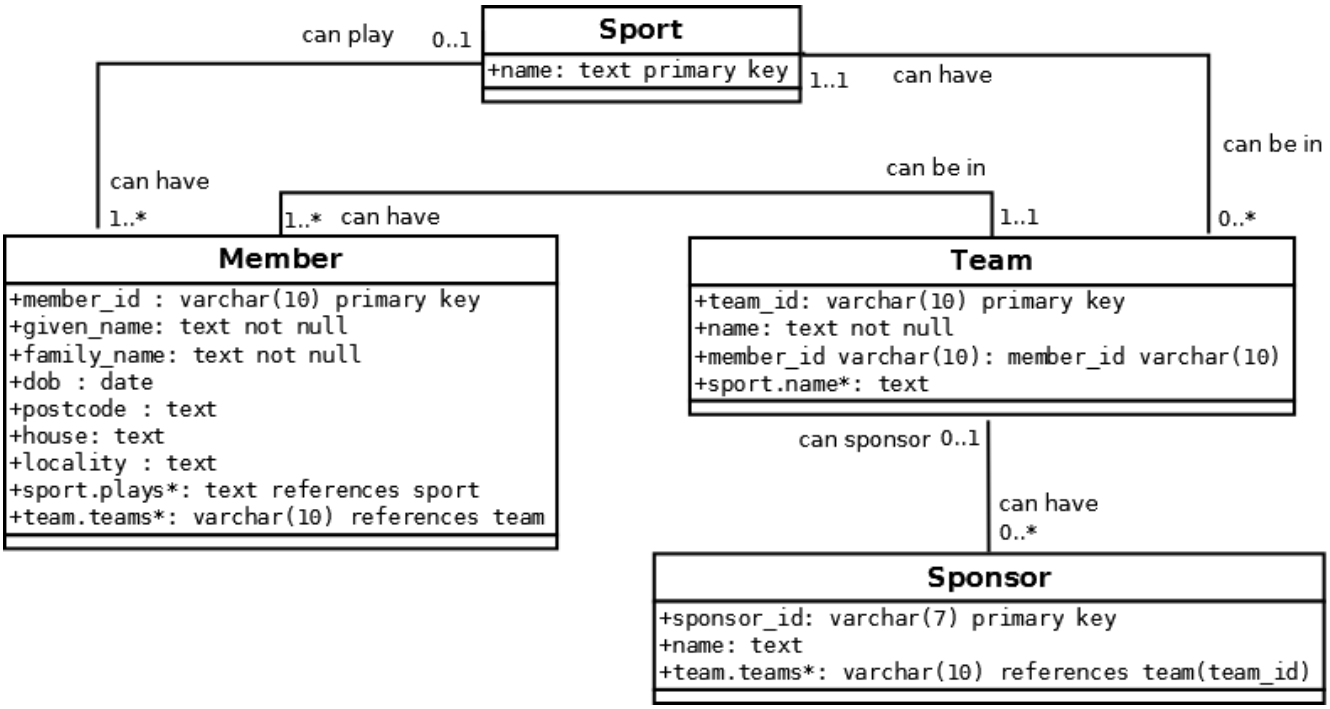

Illustration 1: UML Class Diagram showing relational model implemented by given SQL

This class diagram is unable to complete the requirements of the enterprise rules a fails to capture the relationships shows in the entity-relationship model given in assignment specification. See Appendix A – Illustration 2. There are many problems with this implementation.

The first problem is the fact that it is possible to add duplicates of a member to the database. The reason why this is possible is because the primary key is set to the member_id, and has no other constrains in the table. This will allow for members with exactly the same name and location to be added provided there member_id field follows primary key constraints, this is that a primary key must be unique and not null.

The second problem is that “teams” and “plays” are both foreign keys within the member table should not be in this table as they should be in a link table that is required to break up the many to many relationships that exist. The member_id should also be removed from the team table which need to be placed into a link table to break up the many to many relationships.

There also needs to be some form of trigger or constraint that will activate to prevent members from joining a team which plays a sport that they do not. Doing this will prevent

members from appearing to belong to a team but the team only having one member who belong to the team but not play the sport in question.

The reason why there is a lack of consistency within the database is that there need to be link tables that will break up the many to any relationships as previously explained in the above paragraph.

The reason why a team can have many sponsors is that in the sponsor table there is a foreign key called teams which references team_id. This will allow for each team to have many different sponsors or not have a sponsor because the field has no constraints as to weather it is null or not. Also with this relationship it means that only one sponsor can sponsor a team because the primary key the sponsor_id.

Correct Implementation

Below is a UML Class Diagram showing the correct way to implement the entity-relationship diagram and enterprise rules, located in Appendix A.

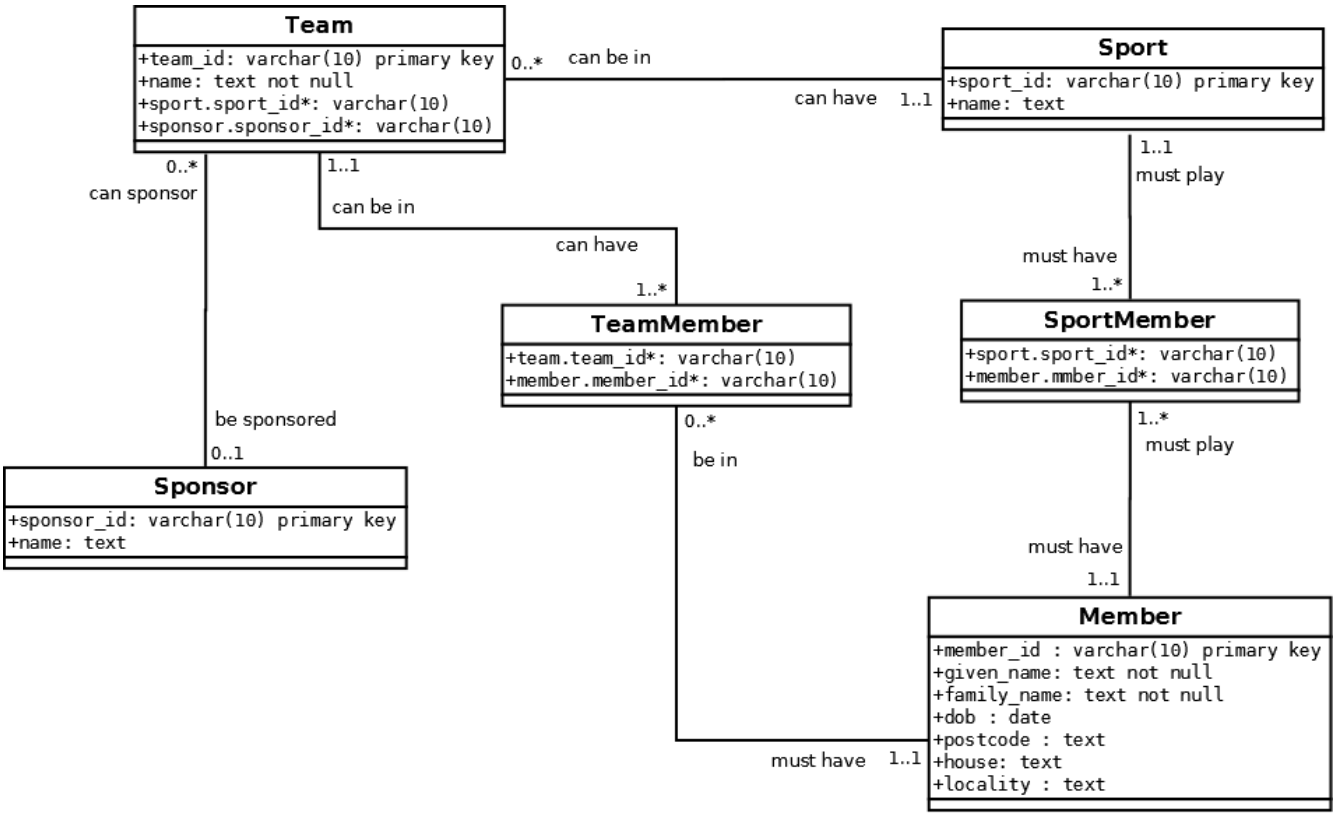

Illustration 2: UML Class Diagram showing the correct implementation

The sponsor table has two fields these are sponsor_id and name. Sponsor_id is a primary key and is used as a foreign key in the team table. In the team table the sponsor_id can be set as null which means that it is possible for a team to not have a sponsor. Due to the fact that the sponsor_id is a primary key each team can only have one sponsor.

The team table has two foreign keys from tables sponsor and sport. Its primary key is the team_id field. This is used a foreign key in the link table TeamMember. This allows for one or many teams to be played by one or more members. The team table also has a sport foreign key that ensures that a team has to have one and only one sport.

The sport table has two fields one a primary key and another of type text called name. The primary key is called sport_id, this is used as a foreign key in team and SportMember. This makes sure that each sport can have one or more members and ensures that Every sport is played by at least one club member.

The table member has one primary key called member_id and is used in two other tables as foreign keys

TeamMember and SportMember. This insures that a member may join a team an that each team has at least one member. These constrains are not sufficient to avoid all problems. One such problem is that when adding a new player to a team they must play the sport in question. This has not been implemented using primary and foreign key rules.

SQL Commands

These are the commands that have been used to create the correct implementation of the UML Class Diagram.

```
CREATE TABLE `sponsor`
(
	sponsor_id varchar(10) NOT NULL,
	name TEXT NOT NULL,
	PRIMARY KEY(sponsor_id)
)ENGINE=InnoDB;


CREATE TABLE `sport`
(
	sport_id varchar(10) NOT NULL,
	name TEXT NOT NULL,
	PRIMARY KEY(sport_id)
)ENGINE=InnoDB;


CREATE TABLE `team`
(
	team_id varchar(10) NOT NULL,
	name TEXT NOT NULL,
	sport_id varchar(10) NOT NULL,
	sponsor_id varchar(10),
	PRIMARY KEY(team_id),
	FOREIGN KEY(sport_id) REFERENCES sport(sport_id),
	FOREIGN KEY(sponsor_id) REFERENCES sponsor(sponsor_id)
)ENGINE=InnoDB;


CREATE TABLE `member`
(
	member_id varchar(10) NOT NULL,
	given_name varchar(30) NOT NULL,
	family_name varchar(50) NOT NULL,
	dob DATE NOT NULL,
	postcode varchar(8) NOT NULL,
	house varchar(50) NOT NULL,
	locality varchar(30) NULL,
	CONSTRAINT member_composite PRIMARY KEY(member_id, given_name,
family_name, dob, postcode),
	CONSTRAINT uc_member_composite UNIQUE(given_name, family_name, dob,
postcode)
```

```
)ENGINE=InnoDB;


CREATE TABLE `team_member`
(
	team_id varchar(10) NOT NULL,
	member_id varchar(10) NOT NULL,
	CONSTRAINT team_member_id PRIMARY KEY(team_id, member_id),
	FOREIGN KEY(team_id) REFERENCES team(team_id),
	FOREIGN KEY(member_id) REFERENCES member(member_id)
)ENGINE=InnoDB;


CREATE TABLE `sport_member`
(
	sport_id varchar(10) NOT NULL,
	member_id varchar(10) NOT NULL,
	CONSTRAINT team_member_id PRIMARY KEY(sport_id, member_id),
	FOREIGN KEY(sport_id) REFERENCES sport(sport_id),
	FOREIGN KEY(member_id) REFERENCES member(member_id)
)ENGINE=InnoDB;
```

It is possible to create a MySQL trigger that can be used to check if the member does play the sport in question this would make sure that only members that play the sport in question can be added to the team. Seeing as MySQL does not support this unless you write a trigger hack that will enable this constraint I have decided not to implement this and to include the code that I had got too before I found this out.

```
delimiter |

CREATE TRIGER check_member_sport BEFORE INSERT ON team_member
	FOR EACH ROW BEGIN
		DECLARE sport_tmp varchar(10);
		SELECT sport_id INTO sport_tmp FROM sport_member WHERE
member_id=new.member_id; – Some SQL that will check
		IF ( sport_tmp IS NULL ) THEN
			call fail('It should not insert!');
		END IF;
	END;|

delimiter ;
```

I had to make sure that the database engine that the system used to store the tables was specificity InnoDB. InnoDB provides me with fully working primary and foreign key constrains, where as the default engines allows violations of the constraints.

SQL Testing

Below you will find tests that show that some of the problems described in the assignment are avoided.

Problem 1

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house, locality)
VALUES('ME01', 'Pete', 'Maynard', '1989-08-09', 'SA43 2RU', '1 Penual Cottage', 'Pembs');

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house, locality)
VALUES('ME02', 'Scott', 'Jones', '1991-12-25', 'SY23 1PZ', '17 Bridge Street', 'Ceridigeon');

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house, locality)
VALUES('ME03', 'Scott', 'Jones', '1964-02-20', 'SY23 1PZ', '17 Bridge Street', 'Ceridigeon');

This creates a three users two of which live in the same house and have the same name but they have a different date of birth. This is find as the fields are still unique. It is not possible to do the following.

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house, locality)
VALUES('ME01', 'Scott', 'Jones', '1991-12-25', 'SY23 1PZ', '17 Bridge Street', 'Ceridigeon');

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house, locality)
VALUES('ME02', 'Scott', 'Jones', '1991-12-25', 'SY23 1PZ', '17 Bridge Street', 'Ceridigeon');

Problem 2 and 5

This is the SQL to add users to the team table. First you need to create teams to add members too, as well as the sponsors for the teams. Below creates two teams one with a sponsor and one with out, it also creates a sponsor.

INSERT INTO `sponsor` (sponsor_id, name) VALUES('SN01', 'Cambrian Hotel');

INSERT INTO `team` (team_id, name, sport_id, sponsor_id) VALUES('TM01', 'Welsh Karate Kai', 'SP01', NULL);
INSERT INTO `team` (team_id, name, sport_id, sponsor_id) VALUES('TM02', 'Kendo Kai', 'SP02', 'SN01');

Next we need to add members to the team by, like so:

INSERT INTO `team_member` (team_id, member_id) VALUES ('TM01', 'ME01');

This adds member “ME01” to team “TM01”. It is not possible to add a user to a team if they do not exists. The following will fail.

INSERT INTO `team_member` (team_id, member_id) VALUES ('TM01', 'ME453');

Problem 3

This is the SQL that joins members to the sport in which they play.

INSERT INTO `sport_member` (sport_id, member_id) VALUES ('SP01', 'ME01');
INSERT INTO `sport_member` (sport_id, member_id) VALUES ('SP02', 'ME02');

This will make sure that sport 'SP01' is played by member 'ME01'. It is not possible to make members play sports that do not exist and vice versa. See below for error.

INSERT INTO `sport_member` (sport_id, member_id) VALUES ('SP501', 'ME32');

Problem 4

To make sure that a member is unable to join a team in which they do not play the sport in question has not been implemented, there is a basic trigger in the SQL code that might would have been able to prevent this. But as MySQL does not officially support this sort of constrains I have not written a trigger hack to create this constraint.

Appendix A

SQL From assignment

SQL commands used in the creation of the UML Class Diagram showing relational model implemented by given SQL. These are from the assignment specification.

```
CREATE TABLE sport (
        name text primary key
);

CREATE TABLE team (
        team_id varchar(10) primary key,
        name text not null,
        member_id varchar(10),
        sport text references Sport (name)
);

CREATE TABLE member (
        member_id varchar(10) primary key,
        given_name text not null,
        family_name text not null,
        dob date,
        postcode text,
        house text,
        locality text,
        plays text references sport,
        teams varchar(10) references team
);

CREATE TABLE sponsor (
        sponsor_id varchar(7) primary key,
        name text,
        teams varchar(10) references team (team_id)
);
```

Enterprise Rules

These are the enterprise rules that were used to create the correct implementation, these are from the assignment specification.

"*Every member of Llandwp Sports club plays at least one sport; for example, Fred Jones plays soccer,rugby and badminton, and Joanna Taylor plays table tennis. Every sport is played by at least one club member.*

Some sports have one or more teams, but a sport need not have any team – some sports are

purely recreational. Each team is for one and only one sport.

Every team has at least one player and may have more. A player on a team must be a club member, and must play the sport in question. A club member may belong to no teams, or to one team, or to several teams.

A team may have at most one sponsor. A sponsor sponsors zero or more teams."

Entity-relationship model

This is the entity-relationship model from the assignment specification:

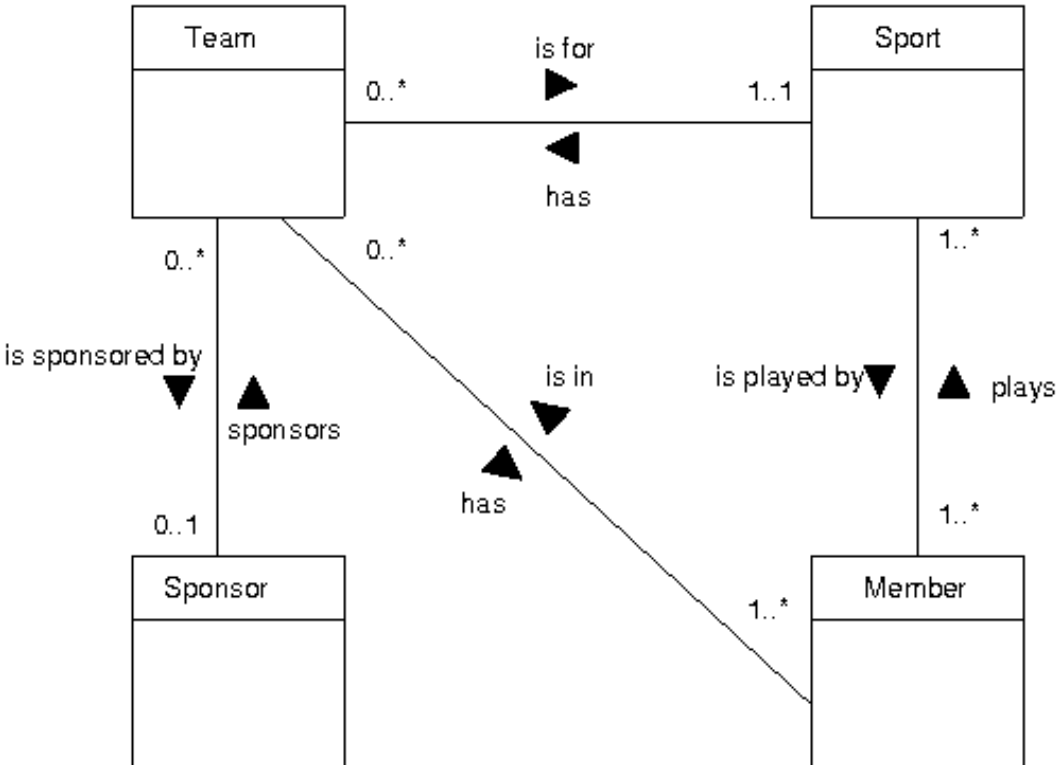

Illustration 3: Entitiy-relationship model from assignment specification

Complete SQL file

```
-- Llandwp Sport Club
-- CS27020 Assignment
-- Peter Mayanrd

CREATE TABLE `sponsor`
(
	sponsor_id varchar(10) NOT NULL,
	name TEXT NOT NULL,
	PRIMARY KEY(sponsor_id)
)ENGINE=InnoDB;

-- INSERT DATA
INSERT INTO `sponsor` (sponsor_id, name) VALUES('SN01', 'Cambrian Hotel');
INSERT INTO `sponsor` (sponsor_id, name) VALUES('SN02', 'Port Twenty 2');

CREATE TABLE `sport`
(
	sport_id varchar(10) NOT NULL,
	name TEXT NOT NULL,
	PRIMARY KEY(sport_id)
)ENGINE=InnoDB;

-- INSERT DATA
INSERT INTO `sport` (sport_id, name) VALUES('SP01', 'Karate');
INSERT INTO `sport` (sport_id, name) VALUES('SP02', 'Kendo');

CREATE TABLE `team`
(
	team_id varchar(10) NOT NULL,
	name TEXT NOT NULL,
	sport_id varchar(10) NOT NULL,
	sponsor_id varchar(10),
	PRIMARY KEY(team_id),
	FOREIGN KEY(sport_id) REFERENCES sport(sport_id),
	FOREIGN KEY(sponsor_id) REFERENCES sponsor(sponsor_id)
)ENGINE=InnoDB;

-- INSERT DATA
INSERT INTO `team` (team_id, name, sport_id, sponsor_id) VALUES('TM01', 'Welsh Karate
Kai', 'SP01', NULL);
INSERT INTO `team` (team_id, name, sport_id, sponsor_id) VALUES('TM02', 'Kendo Kai',
'SP02', 'SN01');

CREATE TABLE `member`
```

```
(
	member_id varchar(10) NOT NULL,
	given_name varchar(30) NOT NULL,
	family_name varchar(50) NOT NULL,
	dob DATE NOT NULL,
	postcode varchar(8) NOT NULL,
	house varchar(50) NOT NULL,
	locality varchar(30) NULL,
	CONSTRAINT member_composite PRIMARY KEY(member_id, given_name,
family_name, dob, postcode),
	CONSTRAINT uc_member_composite UNIQUE(given_name, family_name, dob,
postcode)
)ENGINE=InnoDB;

-- INSERT DATA
INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house,
locality)
	VALUES('ME01', 'Pete', 'Maynard', '1989-08-09', 'SA43 2RU', '1 Penual Cottage',
'Pembs');

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house,
locality)
	VALUES('ME02', 'Scott', 'Jones', '1991-12-25', 'SY23 1PZ', '17 Bridge Street',
'Ceridigeon');

INSERT INTO `member` (member_id, given_name, family_name, dob, postcode, house,
locality)
	VALUES('ME03', 'Scott', 'Jones', '1964-02-20', 'SY23 1PZ', '17 Bridge Street',
'Ceridigeon');

--
-- LINK TABLES
--

CREATE TABLE `team_member`
(
	team_id varchar(10) NOT NULL,
	member_id varchar(10) NOT NULL,
	CONSTRAINT team_member_id PRIMARY KEY(team_id, member_id),
	FOREIGN KEY(team_id) REFERENCES team(team_id),
	FOREIGN KEY(member_id) REFERENCES member(member_id)
)ENGINE=InnoDB;

-- TRIGERS
-- When new member is added to team, check that member plays team.sport
/*
delimiter |
```

```
CREATE TRIGER check_member_sport BEFORE INSERT ON team_member
	FOR EACH ROW BEGIN
		-- DECLARE sport_tmp varchar(10);
		SELECT sport_id INTO sport_tmp FROM sport_member WHERE
member_id=new.member_id;
		IF ( sport_tmp IS NULL ) THEN
			call fail('It should not insert!');
		END IF;
	END;|

delimiter ;
*/
-- INSERT DATA
INSERT INTO `team_member` (team_id, member_id) VALUES ('TM01', 'ME01');
INSERT INTO `team_member` (team_id, member_id) VALUES ('TM01', 'ME02');

CREATE TABLE `sport_member`
(
	sport_id varchar(10) NOT NULL,
	member_id varchar(10) NOT NULL,
	CONSTRAINT team_member_id PRIMARY KEY(sport_id, member_id),
	FOREIGN KEY(sport_id) REFERENCES sport(sport_id),
	FOREIGN KEY(member_id) REFERENCES member(member_id)
)ENGINE=InnoDB;

-- INSERT DATA
INSERT INTO `sport_member` (sport_id, member_id) VALUES ('SP01', 'ME01');
INSERT INTO `sport_member` (sport_id, member_id) VALUES ('SP02', 'ME02');
```