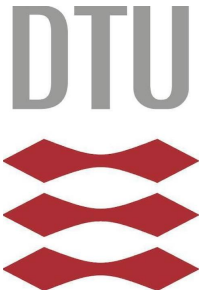

Technical University of Denmark

02170 - Database Systems

Group: T1

Creation and Implementation of a Database for the Olympics

Author:	*Student Nr.:*
Nicolai Herrmann	s164438
Capucine Duclos	s171751
Emma Demarecaux	s176437

April 18, 2018

Contents

1 Statement of requirements **2**

2 Conceptual design **3**

3 Logical design **5**

4 Normalization **7**

5 Implementation **8**

6 Database instance **9**

7 SQL Data Queries **12**

8 SQL Table Modifications **14**

9 SQL Programming **16**
9.1 Functions 16
9.2 Procedures 16
9.3 Events 17
9.4 Triggers 17
9.5 Transactions 18

1 Statement of requirements

In this section is explained how the problem has been modelled.

The Olympic games consists of several Tournaments. Each tournament hosts a number of Matches, and is played in a given Sport. Tournaments same sex only. Each tournament is played during a specific period that starts at a StartDate and ends at an EndDate. Each tournament ends with a podium of 3 Team winners.

Each Macth must have at most 3 Team winners, and lasts a given period starting from a StartDate and ending at EndDate.

Each Team consists of a number of Participants and is competing for a specific Country. Teams must participate in at least one Macth. After a tournament, a team can earn a gold, a silver or a bronze medal and the Results of total number of gold, silver and bronze medals must be recorded for each team.

Each Participant must belong to at least one Team: this means that he or she can compete individually (a team of 1) and/or with other participants. Characteristics of particpants must be recorded such as their name, birthday, sex, height, weight and what Country they compete for. A participant doesn't have to compete for a country, if for example the country is not recognized by the United Nation or if it undergoes political issues. In this case, the participant will compete under the Olympics banner.

Each Country can be referred to with its Initials, or its entire Name. Also, during the Olympics, each country earns a number of gold, silver and bronze medals.

Each Sport has a Name, and belongs to a sport Category. Also each Category has its own Name, and covers at least one Sport

2 Conceptual design

From the statement of requirements explained earlier, a conceptual model has been created, and depicted in the figure 2.1 below. Description of the relations between entities are described below.

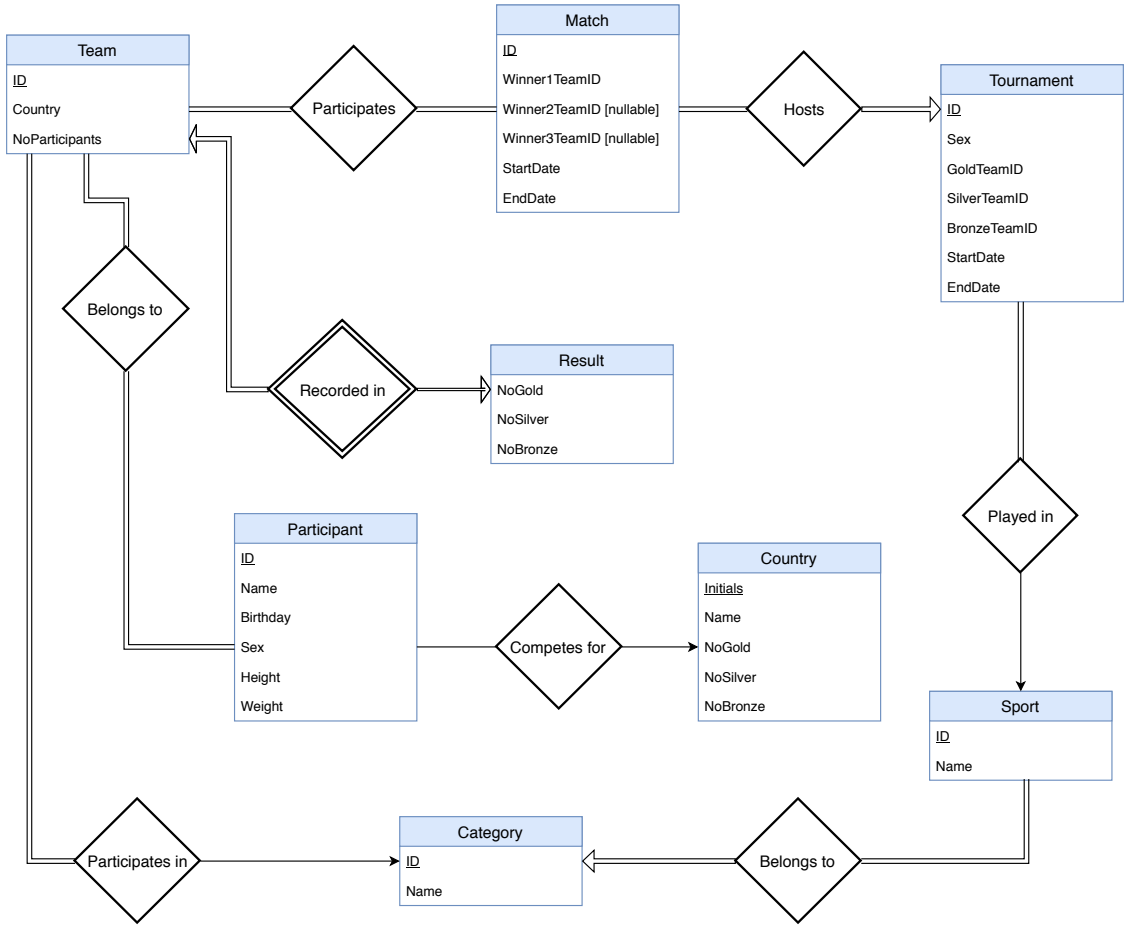

Figure 2.1: Conceptual model

Tournament - Sport
Starting from the top, we defined a sport in the tournament. Instead, we could have defined it in the matches which makes up the tournament. Since matches in tournaments is a many to one relationship, the information would be repeated more than necessary. Each tournament can only have one sport and therefore has a one to one relation.

Sport - Category
Each sport belongs to a category, which is the type of sport, e.g. water sports. The category enitity is not strictly necessary, but was chosen to be included, since it can

give a nice separation of the sports. It also provides a simple way to define the types of sports that teams can participate in.

Team - Category
It is useful to know what sports a team participates in, so each team has a category of sports they can participate in. Alternatively there could be a table connecting teams to sports, but since teams will always participate in sports within the same category, it would be a waste to list them separately.

Participant - Country
Each participant may or may not have a country they compete for. Indeed, some may choose to be independent,for instance due to political reasons, and compete under the Olympics banner. This is why we have a one to one relation between Participant and Country.

Team - Participants
Each Team can be composed of one or many participants that will participate in matches. It is also possible to have two teams with the same participants, if they participate in more than one category, as long as the TeamID is different. Each team should have at least one participant and each participant should be in at least one team as it is possible for a team to consists of a single person.

Tournament - Match
A tournament consists of two types of matches, one which has only one winner team e.g. soccer match, and one which has a podium of three top teams e.g. swimming. We have one match table, which supports both cases by having information of the top three teams, with team 2 and 3 being nullable, i.e. don't need to be set. Alternatively, two different match tables could be created, one for each type of match, but this would of course require more tables.

Match - Team
Each match has a list of participating teams, which alternatively could be listed in the match table itself. This would require a dedicated column for each team, which in term would limit the amount of participating teams. This is why we chose to have a dedicated table for the participants.

Team - Results
The results for all tournaments for all teams should be recorded. The result of a tournament is already recorded in the tournament table, which allows one to retrieve the results for a team by searching the tournaments. We chose to have a table which is always updated, that keeps track of teams records. The table is initialized with an entry for every team, and every time a tournament is finished, the table is updated.

3 Logical design

The conceptual model has been transformed into a logical model, as shown in Figure 3.1 below.

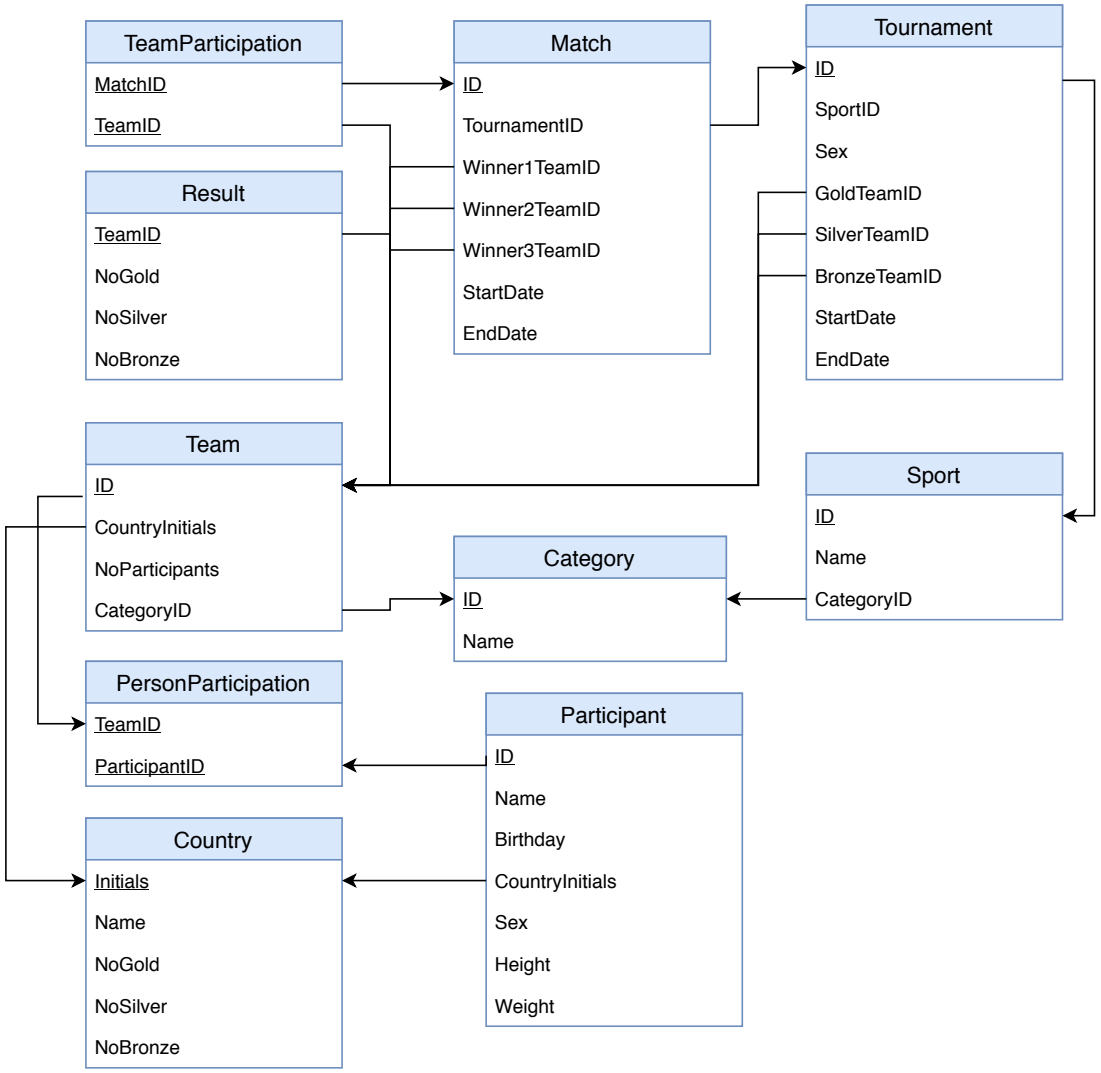

Figure 3.1: Logical model

For many to many relations, it is natural to create a distinct table reefering to the connection, that is the relation between two entities. For instance, a table `TeamParticipation` has been created for the relation `Participates` between the entities `Team` and `Match`. This table shows which teams will participate in which matches thanks to the two attributes `TeamID` and `ParticipantID`. These two attributes therefore respectively constitute foreign keys referencing the tables `Team` and `Participant`.

For many to one relations, it is natural to put the key of the "one" table into the

"many" table. For example, the ID of the entity `Category` has been put into the `Team` table as an attribute. This attribute is a foreign key referencing the `Category` table.

There are no one to one relations in this database. However, the process is the same as in the many to one relations: the concptual model would have been converted into the logical model by putting the key of the "one" table into the other table.

4 Normalization

In this section we investigate whether our tables are normalized or not.

First Normal Form

The value of each cell, in every table of our database, is atomic, which means that it corresponds to a single value. Hence, all our tables are 1NF.

Second Normal Form

Furthermore, in all the tables of our database, the primary key consists either of only one attribute, or, of all the attributes in the relation. The repartition is shown below.

Tables whose primary key is only one attribute:

- Match
- Tournament
- Sport
- Category
- Country
- Participant
- Result
- Team

Tables whose primary key is all attributes:

- Person participation
- Team participation

Hence, all our tables are also 2NF.

Third Normal Form

Each non primary key depends directly on the entire primary key for each table. It doesn't depend transitively via other attributes. Therefore, all our tables are 3NF.

5 Implementation

First, we needed to create our database : `Olympicsdb`.

Tables implementation

Then, in this same database, we created each table individually. The related codes can be seen in appendices. However, it is worth mentionning some particular points in the attributes definitions.

- For the definition of the primary key in the relations `Team, Match, Tournament, Sport, Category, and Partcipant`, we used the command `INT NOT NULL AUTO INCREMENT`, that automatically generates values for the primary keys for each data object. The keys are automatically incremented integers, which result in unique, non zero, keys for every entry.
- For the Country relation, we defined the type of the attribute `Initials` as `varchar (2)`, which means that there can be only two letters maximum for the definition of the initials of any country.
- In the `Sport` relation, the foreign key `CategoryID` refers to `ID` of the `Category` relation. As many sports can be defined for one category, if a given category is deleted, then all the corresponding sports must be deleted as well. This is why we used the `ON DELETE CASCADE` in the definition of the `CategoryID` foreign key. Also, we use the command `ON UPDATE CASCADE` that allows a proper update the `CategoryID` in the `Sport` relation if the `ID` of the `Category` table changes.
- The primary keys of relations `PersonParticipation` and `Team participations` are defined with all the attributes composing the tables. Also, each attribute is also a foreign key to another table. Hence, when defining the foreign keys, the command `ON DELETE CASADE` and `ON UPDATE CASCADE` must be added.

Views implementation

We also implemented views in our database. Thus, the `Female` and `Male` views have been created. These views respectively select the female and the male participants, showing their name and the initials of the country they compete for.

6 Database instance

After implementing the tables and views, we popultated them with new data objetcs. To do this, we used the MySQL `INSERT` command.

N.B.: We inserted some data objects for each tables and we are aware that the number of data objects is not realistic. In the olympics there would be a lot more objects, since there are a lot more teams, sport, etc. However, we believe that our tables provide the necessary functionality and demonstrate that the tables sizes can be expanded greatly and therefore facilitate the actual Olympics.

The results are shown in the tables below.

ID	Name	Birthday	CountryInitials	Sex	Height	Weight
1	Morten	1995	DK	M	175	80.0
2	Elisa	1994	DK	F	174	59.0
3	Sofie	1996	DK	F	173	65.0
4	Anna	1996	DK	F	173	65.0
5	Capucine	1996	FR	F	168	60.0
6	Emma	1996	FR	F	175	65.0
7	Charles	1996	FR	M	175	65.0
8	Sandra	1996	FR	F	175	65.0
9	Karl	1995	SE	M	175	70.0
10	Olivia	1995	SE	F	175	60.0
11	Malte	1995	DE	M	175	70.0
12	Tobias	1995	DE	M	175	70.0
13	Maria	1995	GR	F	175	60.0

Figure 6.1: Table Participant

ID	Name	CategoryID
1	50 metres freestyle	1
2	100 metres freestyle	1
3	4 x 100 metres freestyle relay	1
4	Artistic	2
5	Rhytmic	2
6	Trampoline	2
7	Vollevball (beach)	3
8	Vollevball (indoor)	3

Figure 6.2: Table Sport

ID	Name
1	Swimming
2	Gymnastic
3	Vollevball

Figure 6.3: Table Category

ID	CountryInitials	NoParticipants	CategoryID
1	DK	1	1
2	SE	1	1
3	DE	1	1
4	DK	3	3
5	FR	3	3
6	GR	1	2
7	SE	1	2
8	FR	1	2
9	DE	1	2

Figure 6.4: Table Team

Initials	Name	NoGold	NoSilver	NoBronze
DE	Germanv	0	1	2
DK	Denmark	3	0	0
FR	France	1	1	0
GR	Greece	1	0	0
SE	Sweden	0	3	0

Figure 6.5: Table Country

ID	SportID	Sex	GoldTeamID	SilverTeamID	BronzeTeamID	StartDate	EndDate
1	1	M	1	2	3	2018-07-14 10:00:00	2018-07-14 12:00:00
2	2	M	1	2	3	2018-07-15 10:00:00	2018-07-15 12:00:00
3	7	F	4	5	NULL	2018-07-14 08:00:00	2018-07-14 16:00:00
4	4	F	6	7	NULL	2018-07-16 14:00:00	2018-07-16 18:00:00
5	6	M	8	9	NULL	2018-07-16 08:00:00	2018-07-16 12:00:00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6.6: Table Tournament

ID	TournamentID	Sex	Winner1TeamID	Winner2TeamID	Winner3TeamID	StartDate	EndDate
1	1	M	1	2	3	2018-04-14 10:00:00	2018-04-14 18:13:00
2	2	M	1	2	3	2018-07-15 10:00:00	2018-07-15 12:00:00
3	3	F	4	5	NULL	2018-07-14 08:00:00	2018-07-14 10:00:00
4	3	F	4	5	NULL	2018-07-14 14:00:00	2018-07-14 16:00:00
5	4	F	6	7	NULL	2018-07-16 14:00:00	2018-07-16 18:00:00
6	5	M	8	9	NULL	2018-07-16 08:00:00	2018-07-16 12:00:00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6.7: Table Match

TeamID	NoGold	NoSilver	NoBronze
1	2	0	0
2	0	2	0
3	0	0	2
4	1	0	0
5	0	1	0
6	1	0	0
7	0	1	0
8	1	0	0
9	0	1	0

Figure 6.8: Table Result

We can notice that in the tables of figures 6.5 6.6, 6.7 and 6.8, all numbers of medals are set to zero (or `NULL` values). Indeed, we considered that tables are populated prior to the Olympic games, and thus, the results for each match is not known in advance.

MatchID	TeamID
1	1
1	2
1	3
2	1
2	2
2	3
3	4
3	5
4	4
4	5
5	6
5	7
6	8
6	9

Figure 6.9: Table TeamParticipation

TeamID	ParticipantID
1	1
4	2
4	3
4	4
5	5
5	6
8	7
5	8
2	9
7	10
3	11
9	12
6	13

Figure 6.10: Table PersonParticipation

Name	CountryInitials
Morten	DK
Charles	FR
Karl	SE
Malte	DE
Tobias	DE

Figure 6.11: View Male

Name	CountryInitials
Elisa	DK
Sofie	DK
Anna	DK
Capucine	FR
Emma	FR
Sandra	FR
Olivia	SE
Maria	GR

Figure 6.12: View Female

7 SQL Data Queries

In this section, examples of typical SQL data queries are given.

- **How many gold medals did Charles get ?**

```
SELECT P.Name, SUM(NoGold) AS Number_Gold_Medals
FROM Participant AS P
JOIN Result AS R
JOIN PersonParticipation AS PP
WHERE R.TeamID = PP.TeamID
AND P.ID = PP.ParticipantID
AND P.Name = 'Charles';
```

This query selects the total number of gold medals got by all the teams Charles belongs to. It displays the this result as weel as its name, Charles. The result is shown below.

	Name	Number_Gold_Medals
	Charles	1

Figure 7.1: Charles' medals

- **What is the number of participants for the Scandinavian countries (Denmark and Sweden) ?**

```
SELECT COUNT(p.ID) AS Number_Participants, c.Name
FROM Participant p JOIN Country c
WHERE c.Initials = p.CountryInitials GROUP BY c.Initials HAVING
c.Initials IN ('DK', 'SE');
```

This query counts all the particpants competing for Denmark and Sweden.The result is shown below.

	Number_Participants	Name
	4	Denmark
	2	Sweden

Figure 7.2: Scandinavian participants

- **What is the ranking of countries order by decreasing total number of medals ?**

```
SELECT Name, NoGold + NoSilver + NoBronze AS Number_Medals
FROM Country
GROUP BY Name
ORDER BY Number_Medals DESC;
```

This query gives a ranking of all countries competing in the Olympics. The result is shown below.

	Name	Number_Medals
	Germany	3
	Sweden	3
	Denmark	3
	France	2
	Greece	1

Figure 7.3: Results

8 SQL Table Modifications

In this section, examples of Table Modifications that can be applied in the Olympicsdb are given. For instance, if one participant has broken his arm before the Olympics, he can't participate in the games anymore. Thus, he should be removed from the Participant table, and replaced by another one from his country.

For example, Morten broke his arm before the Olympics and can't participate to the swimming competition. Then we should remove him from the Participant table:

```
DELETE FROM Participant WHERE Name='Morten';
```

He is therefore replaced immediately by Nicolai from the same country (Denmark) and who is going to do the same matches. We should add Nicolai to the `Participant` table:

```
INSERT INTO Participant (Name, Birthday, CountryInitials, Sex,
Height, Weight) VALUES ('Nicolai', 1995, 'DK', 'M', 180, 70.0);
```

ID	Name	Birthday	CountryInitials	Sex	Height	Weight
2	Elisa	1994	DK	F	174	59.0
3	Sofie	1996	DK	F	173	65.0
4	Anna	1996	DK	F	173	65.0
5	Capucine	1996	FR	F	168	60.0
6	Emma	1996	FR	F	175	65.0
7	Charles	1996	FR	M	175	65.0
8	Sandra	1996	FR	F	175	65.0
9	Karl	1995	SE	M	175	70.0
10	Olivia	1995	SE	F	175	60.0
11	Malte	1995	DE	M	175	70.0
12	Tobias	1995	DE	M	175	70.0
13	Maria	1995	GR	F	175	60.0
14	Nicolai	1995	DK	M	180	70.0

Figure 8.1: New Participant table

As a team is defined by its participants and the category they are involved in, we should chagne the team ID for Denmark in the swimming category to indicate that the team has been changed:

```
UPDATE Team SET ID = 10 WHERE ID = 1;
```

Then we make Nicolai involved in that new team for Denmark in the swimming category:

```
INSERT INTO PersonParticipation VALUES (10, 14);
```

The table `TeamParticipation` is automatically updated as `TeamID` is a foreign key who references the `ID` of the `Team` table with the definition : `ON UPDATE CASCADE`. Finally, we update the view of `Male` participants:

```
UPDATE Male SET Name = 'Nicolai' WHERE Name = 'Morten';
```

ID	CountryInitials	NoParticipants	CategoryID
2	SE	1	1
3	DE	1	1
4	DK	3	3
5	FR	3	3
6	GR	1	2
7	SE	1	2
8	FR	1	2
9	DE	1	2
10	DK	1	1

Figure 8.2: New Team table

TeamID	ParticipantID
4	2
4	3
4	4
5	5
5	6
8	7
5	8
2	9
7	10
3	11
9	12
6	13
10	14

Figure 8.3: New PersonParticipation table

MatchID	TeamID
1	2
1	3
1	10
2	2
2	3
2	10
3	4
3	5
4	4
4	5
5	6
5	7
6	8
6	9

Figure 8.4: New TeamParticipation table

Name	CountryInitials
Charles	FR
Karl	SE
Malte	DE
Tobias	DE
Nicolai	DK

Figure 8.5: New View Male

9 SQL Programming

9.1 Functions

Functions are a piece of code which returns a value. It can be used to reduce duplicate code or to make a long query more readable. In our database, we've used a function for calculating a difference, with nullprotection, which can be seen on figure 9.1, and to retrieve a piece of information, which can be seen on figure 9.2.

```
DELIMITER //
CREATE FUNCTION getDifference (oldCount int, newCount int) RETURNS int
BEGIN
if oldCount is null then return newCount; end if;
if newCount is not null then return newCount - oldCount; end if;
return - oldCount;
END; //
DELIMITER ;
```

Figure 9.1: Function with nullprotection in mysql

```
DELIMITER //
CREATE FUNCTION getTeamCountry (TeamID int) returns varchar(2) BEGIN return (select CountryInitials from Team where ID = TeamID);END; //
DELIMITER ;
```

Figure 9.2: Function for retrieving a given country

9.2 Procedures

Procedures are a way to group together functionality which unlike functions don't just do calculations, but actually alter tables. In the database we use a procedure to define the logic, which chooses three winners for a match, which can be seen on figure 9.3.

```
DELIMITER //
CREATE procedure updat ()
BEGIN
declare currentTime varchar(10) default substr(current_timestamp(),1, 10);
update `match` set Winner1TeamID = (select TeamID from teamparticipation where  `match`.ID = teamparticipation.MatchID limit 1)
where currentTime = substr(`Date`, 1, 10);
update `match` set Winner2TeamID = (select TeamID from teamparticipation where  `match`.ID = teamparticipation.MatchID and
teamparticipation.TeamID <> winner1TeamID limit 1) where currentTime = substr(`Date`, 1, 10);
update `match` set Winner3TeamID = (select TeamID from teamparticipation where  `match`.ID = teamparticipation.MatchID and
teamparticipation.TeamID <> winner1TeamID and teamparticipation.TeamID <> winner2TeamID limit 1) where currentTime = substr(`Date`, 1, 10);
END; //
DELIMITER ;
```

Figure 9.3: Procedure to choose winners of a match

9.3 Events

Events can be used to execute code at a given time or interval. In the database we use events to choose the winners of matches by calling the procedure on figure 9.3 from the event which can be seen on figure 9.4. In the current database, 3 pseudo random winners are chosen, but in real life this would instead update by pulling the results of that days matches from another database. It could alternatively be used to check if all the matches of the day had a winner.

```
create event anEvent
on schedule every 1 day starts '2018-04-14 19:10:00'
do call updat();
```

Figure 9.4: Event used to update results of matches

9.4 Triggers

Trigger are used to update data in one table that depends on data in other. In the database we have two tables which are a summation of the results of the tournaments. Everytime the tournament table is updated two triggers are called, one which updates the teams table, which can be seen on figure 9.5, and one which updates the results table, which can be seen on figure 9.6. An alternative use for triggers is to populate tables dependent on entries in other tables. In the database we use a trigger to create an entry in the Result table for every new entry in the Team table, since there should exist an entry in Result for every Team. This trigger can be seen on figure 9.7.

```
DELIMITER //
CREATE TRIGGER Tournament_Result
AFTER update ON Tournament
FOR EACH ROW
BEGIN
    if OLD.GoldTeamID is not null
    then update result
        set NoGold = NoGold - 1 where TeamID = OLD.GoldTeamID;
    end if;
    if OLD.SilverTeamID is not null
    then update result
        set NoSilver = NoSilver - 1 where TeamID = OLD.SilverTeamID;
    end if;
    if OLD.BronzeTeamID is not null
    then update result
        set NoBronze = NoBronze - 1 where TeamID = OLD.BronzeTeamID;
    end if;

    if NEW.GoldTeamID is not null
    then update result
        set NoGold = NoGold + 1 where TeamID = NEW.GoldTeamID;
    end if;
    if NEW.SilverTeamID is not null
    then update result
        set NoSilver = NoSilver + 1 where TeamID = NEW.SilverTeamID;
    end if;
    if NEW.BronzeTeamID is not null
    then update result
        set NoBronze = NoBronze+ 1 where TeamID = NEW.BronzeTeamID;
    end if;
END; //
DELIMITER ;
```

Figure 9.5: Trigger to update the Team table

```
DELIMITER //
CREATE TRIGGER Result_Country
AFTER update ON Result
FOR EACH ROW
BEGIN
    update country set NoGold = NoGold + getDifference(OLD.NoGold, NEW.NoGold) where Initials = getTeamCountry(OLD.TeamID);
    update country set NoSilver = NoSilver + getDifference(OLD.NoSilver, NEW.NoSilver) where Initials = getTeamCountry(OLD.TeamID);
    update country set NoBronze = NoBronze + getDifference(OLD.NoBronze, NEW.NoBronze) where Initials = getTeamCountry(OLD.TeamID);

END; //
DELIMITER ;
```

Figure 9.6: Trigger to update the Result table

```
DELIMITER //
CREATE TRIGGER Team_Result
AFTER INSERT ON Team
FOR EACH ROW
BEGIN
    insert Result VALUES (New.ID, 0, 0,0);
END; //
DELIMITER ;
```

Figure 9.7: Trigger to insert into Result table

9.5 Transactions

Transactions are used to ensure that all queries inside the transactions are successfull. If one of the queries fail, the entire transaction is rolled back so that no changes were

made to the database. During a transactions, the changes are kept in memory, and wont affect the database until the end when they are committed. The act of replacing one participant with another in Participant, can be wrapped in a transaction to ensure that both the add and delete was successfull, which can be seen on figure 9.8. The transaction is a demonstration and only includes the actual delete and insert of Participants. Even though the new person is added before the old one is deleted, the new person will not be added if there is no old person to delete, because the transaction is rolled back. As a result, the transaction can be run multiple times, but will only affect the database once, on the first execution.

```
DELIMITER //
CREATE PROCEDURE replacePerson (in oldPerson varchar(48), newPerson varchar(48))
BEGIN
DECLARE oldAmount int default 0;
START TRANSACTION;
insert into Participant (Name, Birthday, CountryInitials, Sex, Height, Weight) VALUES (newPerson, 1996, 'DK', 'F', 185, 73.0);
SET oldAmount = (select count(*) FROM Participant WHERE `Name`=oldPerson);
DELETE FROM Participant WHERE `Name`=oldPerson;
IF (oldAmount - (select count(*) FROM Participant WHERE `Name`=oldPerson) = 0 )
THEN ROLLBACK;
ELSE COMMIT;
END IF;
END; //
DELIMITER ;
```

Figure 9.8: Transaction of replacing a participant