Revolutionizing Sports Data Management: A Case Study of Migrating from Relational Databases to Knowledge Graphs

INFO9014-1 : KNOWLEDGE REPRESENTATION AND REASONING

Professor

Pr. Christophe DEBRUYNE

Author

Wilfried Mvomo Eto - s226625

Academic year 2023 - 2024

Contents

Astract

This paper presents a case study of migrating a sports organization's relational database to a Knowledge Graph-based system. SportWorld has historically used a relational database to manage football-related data but recognized the limitations of this approach. The migration process involved analyzing the existing database, identifying key entities, and transforming the data into a semantic Knowledge Graph. This transition aims to leverage interconnected data for intuitive exploration, advanced analytics, and knowledge discovery. Benefits include enhanced querying through SPARQL, improved insights into player-team dynamics, and predictive capabilities. The case study highlights the transformative potential of Knowledge Graphs in data management and analysis.

Introduction

Historically dependent on a relational database to manage intricate aspects of football, such as player data, team statistics, and match details, SportWorld recognized the inherent limitations of this approach. In response, the organization embarked on a bold migration journey towards an infrastructure based on Knowledge Graphs. This endeavor aimed not only to address existing challenges but also to unlock the inherent potential of interconnected data models.

The ensuing narrative delineates the meticulous steps involved in this migration process. A meticulous analysis of the structure of the legacy relational database paved the way for identifying key entities, relationships, and attributes that constitute the foundation of the organization's knowledge ecosystem. The transformation unfolds through multiple stages, encompassing data extraction, semantic modeling, establishment of entities and relationships, as well as the seamless integration of external data sources.

By embracing the framework of Knowledge Graphs, SportWorld strives to harness the latent semantic richness of interconnected data. This paradigm shift ushers in an array of possibilities, spanning from intuitive data exploration and sophisticated analysis to serendipitous discoveries of previously hidden insights. The advantages range from enhanced data querying capabilities facilitated by SPARQL to a better understanding of player-team dynamics and even the ability to predict team performance based on historical trends.

The portability of this paradigm across various industries reveals a spectrum of opportunities for improved decision-making, innovation, and holistic knowledge exploration. In the ensuing pages, we delve into the intricate journey undertaken by SportWorld, illuminating the processes, challenges, and transformative outcomes that underpin the migration from a traditional relational database to a Knowledge Graph-based system.

1. Database

i. Purpose of the Database

The goal of the MySQL database in the context of the provided schema is to store and manage data related to top-tier football (soccer) leagues. The database structure is designed to capture information about teams, stadiums, players, managers, referees, and various statistics associated with players and teams competing in premier divisions.

By using this database, you can:

- Store information about continents, countries, cities, and their relationships.
- Maintain a list of teams, including their names, gender (male/female), and association with countries and stadiums.
- Manage player data, including their (respective identities) names, place of birth, date of birth, gender, country (nationality), and team affiliation. In our context, we assume that each player has a single nationality.
- Store information about managers and referees, including their (respective identities) names, place of birth, date of birth, gender, and country (nationality). In our context, we assume that each manager and referee has a single nationality.
- Track player statistics such as goals, assists, yellow cards, and red cards, associated with specific players and dates.
- Track team statistics such as the number of red cards, yellow cards, goals for, and goals against, associated with specific teams and dates.

ii. Conceptual schema

Before venturing into the intricacies of a database design, let's first establish a conceptual schema, outlining the high-level structure and relationships between key entities within our data mode. To maintain clarity and coherence, you may want to continue the description of the remaining tables as follows :

continents:

Stores information about continents. Columns: id (primary key), name : name of the continent.

countries:

Stores information about countries. Columns: id (primary key), name : name of the country, continent_id (foreign key referencing continents.id).

cities:

Stores information about cities. Columns: id (primary key), name : name of the city, country_id (foreign key referencing countries.id).

teams:

Stores information about football teams. Columns: id (primary key), name : name of the team, is_male (specifies if the team is a male team), city_id (foreign key referencing cities.id).

stadiums:

Stores information about stadiums. Columns: id (primary key), name : name of the stadium, team_id (foreign key referencing teams.id), city_id (foreign key referencing cities.id).

players:

Stores information about football players. Columns: id (primary key), first_name : first name of the player, last_name :last name of the player, place_of_birth (foreign key referencing cities.id), birthday, country_id (foreign key referencing countries.id) : player nationality, team_id (foreign key referencing teams.id), is_male : player gender (specifies if the player is a male).

managers:

Stores information about football managers. Columns: id (primary key), first_name :first name of the manager, last_name :last name of the manager, place_of_birth (foreign key referencing cities.id), birthday, country_id (foreign key referencing countries.id) : manager nationality, team_id (foreign key referencing teams.id), is_male: manager gender (specifies if the manager is a male).

referees:

Stores information about referees. Columns: id (primary key), first_name : first name of the referee, last_name :last name of the referee, place_of_birth (foreign key referencing cities.id), birthday, country_id (foreign key referencing countries.id): referee nationality, is_male : referee gender (specifies if the referee is a male).

player_statistics:

Stores statistics of individual players. Columns: id (primary key), player_id (foreign key referencing players.id), statistics_date : match date, goals : the total number of goals scored by a player in a specific match, assists : the total number of assists realized by a player in a specific match, yellow_cards : the total number of yellow cards conceded by a player in a specific match, red_cards : the total number of red cards conceded by a player in a specific match, position: position that the player occupies on the pitch.

team_statistics:

Stores statistics of football teams. Columns: id (primary key), team_id (foreign key referencing teams.id), statistics_date : match date, number_of_red_cards : the total number of red cards conceded by a team in a specific match , number_of_yellow_cards : the total number of yellow cards conceded by a team in a specific match , goals_for: the total number of goals scored by a team in a specific match (it indicates the offensive performance of the team, reflecting the number of goals they have successfully scored), goals_against the total number of goals conceded by a team in a specific match(it indicates the defensive performance of the team, reflecting the number of goals opponents have scored against them).

Relationships:

- One-to-Many Relationship: Continent to Country

- One-to-Many Relationship: Country to City
- One-to-Many Relationship: City to Team
- One-to-Many Relationship: City to Player and Manager and Referee
- One-to-Many Relationship: Team to Stadium
- One-to-Many Relationship: Team to Player and Manager
- One-to-Many Relationship: Player to Player_Statistics
- One-to-Many Relationship: Team to Team_Statistics

iii. Assumptions:

1. The football database is designed to store information about continents, countries, cities, teams, stadiums, players, managers, referees, player statistics, and team statistics.
2. Each continent can have multiple countries, and each country belongs to a single continent.
3. Each country can have multiple cities, and each city belongs to a single country.
4. Each city can have multiple teams, and each team belongs to a single city.
5. Each team can have multiple stadiums, and each stadium belongs to a single team and city.
6. Each player, manager, and referee has a unique ID and belongs to a country and city based on their place of birth.
7. Players and managers are associated with a team, while referees are not associated with any specific team.
8. Player statistics and team statistics are recorded with a specific date and associated with the respective player or team.
9. The database assumes a one-to-many relationship between players and player statistics, as well as between teams and team statistics.

iv. Integrity Constraints and Business Rules:

1. The *continents* table enforces the foreign key constraint with the *countries* table, ensuring that a continent cannot be deleted if there are countries associated with it.
2. The *countries* table enforces the foreign key constraint with the *continents* table, ensuring that a country cannot be deleted if it belongs to a continent.
3. The *cities* table enforces the foreign key constraint with the *countries* table, ensuring that a city cannot be deleted if it belongs to a country.
4. The *teams* table enforces the foreign key constraint with the *cities* table, ensuring that a team cannot be deleted if it belongs to a city.
5. The *stadiums* table enforces the foreign key constraints with the *teams* and *cities* tables, ensuring that a stadium cannot be deleted if it belongs to a team or city.

6. The *players* table enforces the foreign key constraints with the *cities*, *countries*, and *teams* tables, ensuring that a player cannot be deleted if they have associated records in the *player statistics* table and that their place of birth, country, and team exist.

7. The *managers* table enforces the foreign key constraints with the *cities*, *countries*, and *teams* tables, ensuring that a manager cannot be deleted if they have associated records in the *team statistics* table and that their place of birth, country, and team exist.

8. The *referees* table enforces the foreign key constraints with the *cities* and *countries* tables, ensuring that a referee cannot be deleted if they have associated records and that their place of birth and country exist.

9. The *player_ statistics* table enforces the foreign key constraint with the *players* table, ensuring that player statistics cannot be deleted if the associated player does not exist.

10. The *team_ statistics* table enforces the foreign key constraint with the *teams* table, ensuring that team statistics cannot be deleted if the associated team does not exist.

v. Entity relationship diagram and physical schema of a database:

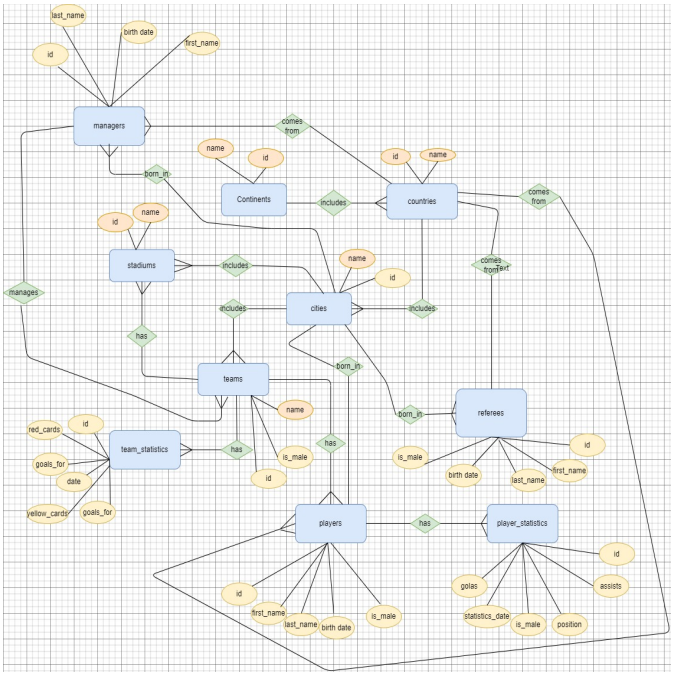

Figure 1: Entity relationship diagram of a database

To transition from a physically centralized design to a semantically centralized one, facilitating the diverse needs of our relational database across departments, ensuring interoperability of semantics is essential. This pivotal phase primarily focuses on crafting an ontology to streamline the retrieval of information within a timely manner. Our aim is to simplify query formulation, eliminating the complexity users previously encountered with the previous relational database. Indeed, the complexity of the queries needed to retrieve meaningful insights can be significant. For instance, consider the following SQL queries:

However, the complexity of the queries needed to retrieve meaningful insights can be significant.

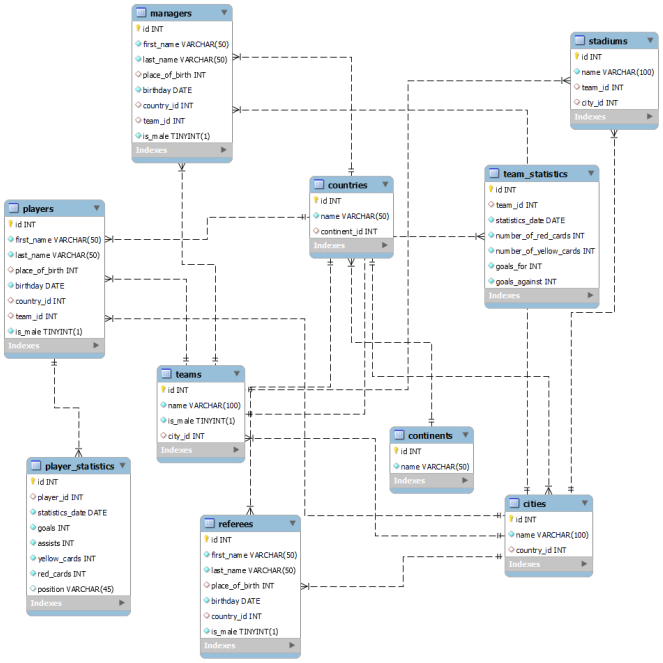

Figure 2: Physical schema of the database

For instance, consider the following SQL queries:

```
/* This query retrieves the first name, last name, and total goals scored by
   players from each continent, providing insights into the performance of players
       from different regions. */
SELECT continents.name AS continent, players.first_name, players.last_name, SUM(
    player_statistics.goals) AS total_goals
FROM players
JOIN countries ON players.country_id = countries.id
JOIN continents ON countries.continent_id = continents.id
JOIN player_statistics ON players.id = player_statistics.player_id
GROUP BY continents.name, players.id
ORDER BY continent, total_goals DESC;
```

Listing 1: Query to retrieve player performance by continent

```
/* This query retrieves the average goals scored by each team, categorized by
    continent,
   providing insights into the performance of teams from different regions. */
SELECT continents.name AS continent, teams.name AS team_name, AVG(team_statistics.
    goals_for) AS avg_goals
FROM teams
JOIN team_statistics ON teams.id = team_statistics.team_id
JOIN countries ON teams.country_id = countries.id
JOIN continents ON countries.continent_id = continents.id
GROUP BY continents.name, teams.id
ORDER BY continent, avg_goals DESC;
```

Listing 2: Query to retrieve team performance by continent

The complexity of these queries illustrates the challenges in extracting comprehensive information from relational databases. To overcome these challenges and enhance data interoperability, it is

advantageous to transform our relational database into RDF (Resource Description Framework). This transformation involves two main steps:

1. Establishing an ontology: This defines the structure and relationships of the data in a semantic format. 2. Creating mappings via R2RML (RDB to RDF Mapping Language): This specifies how data from the relational database is transformed into RDF format.

By doing so, we enable more flexible and powerful querying capabilities through SPARQL, and facilitate better integration with other data sources on the Semantic Web.

2. Ontology

i. Description of the Universe of Discourse

Most often the database is of legacy type but the ontology reflects the semantic concerns regarding the data contents [1]. We introduce the ontology design using Description Logics, which enables us to describe the universe of discourse in terms of concepts, roles, and individuals. The most relevant concepts identified are: **PractitionerIdentity**, **FairPlayIndex**, **Team**, **OriginCountry**, **Location**, **SportsFacility**, and **Penalties**, which are disjoint. Some of them have a subclasses :

- Subclasses of **PractitionerIdentity**: `PlayerIdentity`, `RefereeIdentity` and `ManagerIdentity`;
- Subclasses of **FairPlayIndex**: `FairPlayPlayer` and `FairPlayTeam`;
- Subclasses of **Penalties**: `PlayerPenalties` and `TeamPenalties`;

`FairPlayIndex` represents a category or class related to fair play index within the universe of discourse, `QualificationZone` represents a concept within the ontology that pertains to zones or regions related to qualification in a specific context;`Location` represents a concept within the ontology the place of birth of practitioner and `OriginCountry` represents a concept within the ontology that the native country of practitioner.

The roles are defined as follows:

- $ManagerIdentity$ "coaches" some $Player$;
- $ManagerIdentity$ "managerAttendsIn" some $QualificationZone$;
- $RefereeIdentity$ "refereeAttendsIn" some $QualificationZone$;
- $PlayerIdentity$ "playerAttendsIn" some $QualificationZone$;
- $SportsFacility$ "isLocatedIn" $Location$;
- $FairPlayIndexPlayer$ "gottenBy" some $Player$;
- $FairPlayIndexTeam$ "obtainedBy" some $Team$;
- $PlayerPenalties$ "achievedByPlayer" some $Player$;
- $TeamPenalties$ "achievedByTeam" some $Team$;
- $ManagerIdentity$ "manages" some $Team$.
- $PlayerIdentity$ "playsFor" some $Team$;

- $ManagerIdentity$ "managerIsBornIn" some $Location$;
- $RefereeIdentity$ "refereeIsBornIn" some $Location$;
- $PlayerIdentity$ "playerIsBornIn" some $Location$;
- $ManagerIdentity$ "managerComesFrom" some $Location$;
- $RefereeIdentity$ "refereeComesFrom" some $Location$;
- $PlayerIdentity$ "playerComesFrom" some $Location$;
- $SportsFacility$ "isLocatedIin" some $Location$;
- $SportsFacility$ "belongsToTeam" some $Team$;
- $Team$ "teamIsLocated" some $Location$;
- $Location$ "townOf" some $OriginCountry$;
- $PlayerIdentity$ "playsAtHome" some $Stadium$;

Finally, individuals, for instance, can be $PlayerIdentity(Marcello)$; $ManagerIdentity(Alpha)$; $Location(Ourdjar)$; $OriginCountry(Canada)$; $coaches(Alpha, Marcello)$.

ii. Knowledge base engineering

ALC concept descriptions

On the previous concepts and roles above mentioned which represent respectively classes and properties in OWL, we rely on the ALC for defining the ALC concept descriptions. They are a very important step to ensure that our ontology will be consistent allowing us to realize advanced reasoning. Thereby, we have several concept descriptions among which: PractitionerIdentity, Team, FairPlayIndex, OriginCountry and Location with their subclasses ; ∃ coaches.PlayerIdentity; ∃ manages.Team; ∃ managerAttendsIn.QualificationZone; ∃ isLocatedIn.Location; ∃ gottenBy.FairPlayIndexPlayer. We can form some other ALC concept descriptions by using a complement or intersection, but the goal is to build those which will help to capture the domain of football that we define in the previous milestone.

Terminological Box (TBox) and Assertional Box (ABox)

In the domain of ontology engineering, establishing a comprehensive knowledge base $K =< T, A >$ is essential for capturing the intricacies of a given domain. This knowledge base often comprises a set of statements and relationships that define the fundamental concepts and properties within the domain. In this context, we present a curated set of statements that form the backbone of our ontology, encapsulating various aspects of the domain of interest. Each statement represents a logical assertion about the relationships between entities, providing valuable insights into the structure and dynamics of the domain.

Our knowledge base comprises a generalized terminology T, which is a finite set of the following statements:

1. Every manager identity coaches a player identity:

   $\texttt{ManagerIdentity} \subseteq \exists \texttt{coaches.PlayerIdentity}$.

2. Every Location is town of a origin country: $\texttt{Location} \subseteq \exists\ \texttt{townOf.OriginCountry}$.

3. Every manager identity manages some team : $\texttt{ManagerIdentity} \subseteq \exists \texttt{manages.Team}$.

4. Every referee identity comes from in origin country:

   $\texttt{RefereeIdentity} \subseteq \exists \texttt{refereeComesfrom.OriginCountry}$.

5. Every manager identity comes from in origin country:

   $\texttt{ManagerIdentity} \subseteq \exists \texttt{managerComesfrom.OriginCountry}$.

6. Every player identity comes from in origin country:

   $\texttt{PlayerIdentity} \subseteq \exists \texttt{playerComesfrom.OriginCountry}$.

7. Every manager identity attends in qualification zone:

   $\texttt{ManagerIdentity} \subseteq \exists \texttt{managerAttendsIn.QualificationZone}$.

8. Every player identity plays for some team:

   $\texttt{PlayerIdentity} \subseteq \exists \texttt{playsFor.Team}$.

9. Every player identity attends in some qualification zone:

   $\texttt{PlayerIdentity} \subseteq \exists \texttt{playerAttendsIn.QualificationZone}$

10. Every referee identity attends in qualification zone:

    $\texttt{RefereeIdentity} \subseteq \exists \texttt{refereeAttendsIn.QualificationZone}$.

11. Every sport facility is located in a location:

    $\texttt{SportsFacility} \subseteq \exists \texttt{isLocatedIn.Location}$.

12. Every sport facility belongs to a team:

    $\texttt{SportsFacility} \subseteq \exists \texttt{belongsToTeam.Team}$.

13. Every penalty player is achieved by a player identity:

    $\texttt{PlayerPenalties} \subseteq \exists \texttt{achievedByPlayer.PlayerIdentity}$.

14. Every penalty team is achieved by a team identity:

    $\texttt{TeamPenalties} \subseteq \exists \texttt{achievedByTeam.TeamIdentity}$.

15. Every team is located in a location:

    $\texttt{Team} \subseteq \exists \texttt{teamIsLocated.Location}$.

16. Every player's fair play index is gotten by a player identity:

    $\texttt{FairPlayIndexPlayer} \subseteq \exists \texttt{gottenBy.PlayerIdentity}$.

17. Every team's fair play index is obtained by a team:

    $\texttt{FairPlayIndexTeam} \subseteq \exists \texttt{obtainedBy.Team}$.

All the concepts can be instantiated, thereby the concept satisfiability is done. Besides knowledge base satisfiability is done as well.We verified them by relying on the semantic tableau algorithm. .

iii. Harnessing OWL 2 RL for Scalable Reasoning and Empowering Knowledge Construction

We have selected certain predicates that require first-order logic to be integrated in order to deduce implicit knowledge within a polynomial time frame. To elucidate the importance of **OWL 2 RL** for our ontology and its coherence with the presented knowledge base, we emphasize the crucial role of a robust querying mechanism capable of efficiently managing large datasets.

By employing OWL 2 RL, we augment the specificity and complexity of constraints that govern the properties and relationships among entities within our ontology. Below are exemplar axioms expressible in OWL 2 RL pertinent to our domain:

1. Existential It allows the use of an existential quantification for specifying that instances of one class must have at least one relationship with instances of another class.
2. Constraint on Entity' names: Leveraging OWL 2 RL, we can specify that each Entity must have a unique identifier. This is achieved by employing the hasKey construct, which defines a key for the Identity class comprising the code properties. By doing so, we ensure that no two Identities share the same combination of data type properties identifying identity. Thereby enhancing data integrity and facilitating more precise identification of entities.
3. It enables the use of first-order logic for inferring new knowledge relying on some predicates.

By using OWL 2 RL, we can formally express these constraints and integrate them into our ontology to ensure its consistency and integrity. These constraints help us define clear rules about the structure and relationships of entities in our domain, making it easier to manage and analyze data in our ontology. The base IRIs for the ontology are:

- **TBox**: `http://www.example.org/foot-infos/ont`
  The TBox contains the classes and properties of the ontology.
- **ABox**: `http://www.example.org/foot-infos/resources`
  The ABox contains the instances of the ontology (e.g. cities, players, teams...)

In the process of converting our data to RDF format, we establish several namespace prefixes for clarity and consistency throughout our ontology. These prefixes serve as shorthand references to specific namespaces, simplifying the representation of URIs. Here are the prefixes we'll employ:

- "`foot:`" stands for the ontology namespace, represented by `http://www.example.org/foot-infos/ont`.
- "`res:`" represents the resource namespace, denoted by `http://www.example.org/foot-infos/resources`.

Main Classes: To facilitate the conversion of our Entity-Relationship Diagram (ERD) into RDF format, we establish key structuring classes.The class forms are in PascalCase and delineate the primary entities outlined in the ERD Figure 4 , namely:

- foot:PractitionerIdentity
- foot:Penalties
- foot:FairPlayIndex
- foot:QualificationZone
- foot: Location
- foot:Team
- foot: SportsFacility

These classes symbolize different concepts, hence they are made disjoint. We have the subclasses of some classes mentioned above as well:

For **PractitionerIdentity** class:

- foot:PlayerIdentity
- foot:RefereeIdentity
- foot:ManagerIdentity

For **FairPlayIndex** class:

- foot:FairPlayIndexPlayer
- foot:FairPlayIndexTeam

For **Penalties** class:

- foot:PlayerPenalties
- foot:TeamPenalties

Notice that FairPlayIndexPlayer and FairPlayIndexTeam are disjoint classes.

Properties: The properties associated with our ontology are named using CamelCase convention:

Relation	**Type**
foot:coaches	irreflexive
foot:managerAttendsIn	irreflexive, asymmetric
foot:playerAttendsIn	irreflexive, asymmetric
foot:refereeAttendsIn	irreflexive, asymmetric
foot:playerComesFrom	irreflexive
foot:managerComesFrom	irreflexive
foot:refereeComesFrom	irreflexive
foot:belongsToTeam	irreflexive
foot:achievedByTeam	irreflexive, asymmetric
foot:achievedByPlayer	irreflexive, asymmetric
foot:gottenBy	irreflexive, asymmetric
foot:obtainedBy	irreflexive, asymmetric
foot:isIncludedIn	irreflexive
foot:isLocated	irreflexive
foot:manages	irreflexive
foot:playsFor	irreflexive
foot:townOf	irreflexive
foot:teamIsLocated	irreflexive
foot:cityContinent	irreflexive, asymmetric
foot:teamQualificationZone	irreflexive, asymmetric

There are some inverse object properties like countryConsistsOf (inverse of townOf), playerNativeCountry (inverse of playerIsBornIn), managerNativeCountry (inverse of managerIsBorn), refereeNativeCountry (inverse of refereeIsBornIn),holds, managerEmployer ((inverse of playsFor), playerEmployer

(inverse of `manages`), `isCoachedby` (inverse of `coaches`), `isBorn` (inverse of `comesFrom`), `isMadeUp` and `isIncludedIn`.

Data properties: The properties associated with our ontology are also named using CamelCase convention:

Relation	**Type**	**Data Type**
`foot:playerFirstName`	functional	`xsd:string`
`foot:playerLastName`	functional	`xsd:string`
`foot:playerDateOfBirth`	functional	`xsd:date`
`foot:refereeFirstName`	functional	`xsd:string`
`foot:refereeLastName`	functional	`xsd:string`
`foot:refereeDateOfBirth`	functional	`xsd:date`
`foot:managerFirstName`	functional	`xsd:string`
`foot:managerLastName`	functional	`xsd:string`
`foot:managerDateOfBirth`	functional	`xsd:date`
`foot:playerNativeCountry`	functional	`xsd:string`
`foot:refereeNativeCountry`	functional	`xsd:string`
`foot:managerNativeCountry`	functional	`xsd:string`
`foot:playerGenre`	-	`xsd:boolean`
`foot:managerGenre`	-	`xsd:boolean`
`foot:refereeGenre`	-	`xsd:boolean`
`foot:teamName`	functional	`xsd:string`
`foot:assists`	-	`xsd:int`
`foot:goals`	-	`xsd:int`
`foot:goalsFor`	-	`xsd:int`
`foot:goalsAgainst`	-	`xsd:int`
`foot:cityName`	functional	`xsd:string`
`foot:countryName`	functional	`xsd:string`
`foot:playerYellowCardNumber`	-	`xsd:int`
`foot:playerRedCardNumber`	-	`xsd:int`
`foot:teamYellowCardNumber`	-	`xsd:int`
`foot:teamRedCardNumber`	-	`xsd:int`
`foot:position`	-	`string`
`foot:playerFairPlayDate`	functional	`xsd:date`
`foot:teamFairPlayDate`	-	`xsd:date`
`foot:playerPenaltyDate`	functional	`xsd:date`
`foot:teamPenaltyDate`	functional	`xsd:date`
`foot:qualificationZoneName`	functional	`xsd:string`
`foot:teamQualificationZoneName`	functional	`xsd:string`
`foot:teamCountry`	asymmetric, irreflexive	-

Using `cityName`, `countryName`, `stadiumName`“, `qualificationZoneName`, and `teamName` as key for Location, OriginCountry, SprotsFacility, QualificationZone and Team. This is for uniquely identifying their respective classes, ensuring clear and unambiguous identification within the on-

tology.

Overall, the selection of ‘cityName‘, ‘countryName‘, ‘qualificationZoneName‘, ‘stadiumName‘, and ‘teamName‘ as HasKey properties contributes to a robust and effective ontology design, enhancing clarity, efficiency, and reliability in knowledge representation and reasoning. Those data type properties are literals.Additionally, we need to manually add the range `xsd:date` since it is not included in the range list.

Moreover some data type properties like **cityName**, **countryName**, **foot:teamName**, **foot:stadiumName** and **foot:qualificationZoneName** are designated as "HasKey" attributes. This classification is attributed to their capability to uniquely identify teams, stadiums, countries, cities, and qualifications, respectively, contributing significantly to the dataset's integrity and efficiency in data retrieval and management.

Thanks to "Protégé", an OntoGraph can be used to visualize the process of building an ontology :

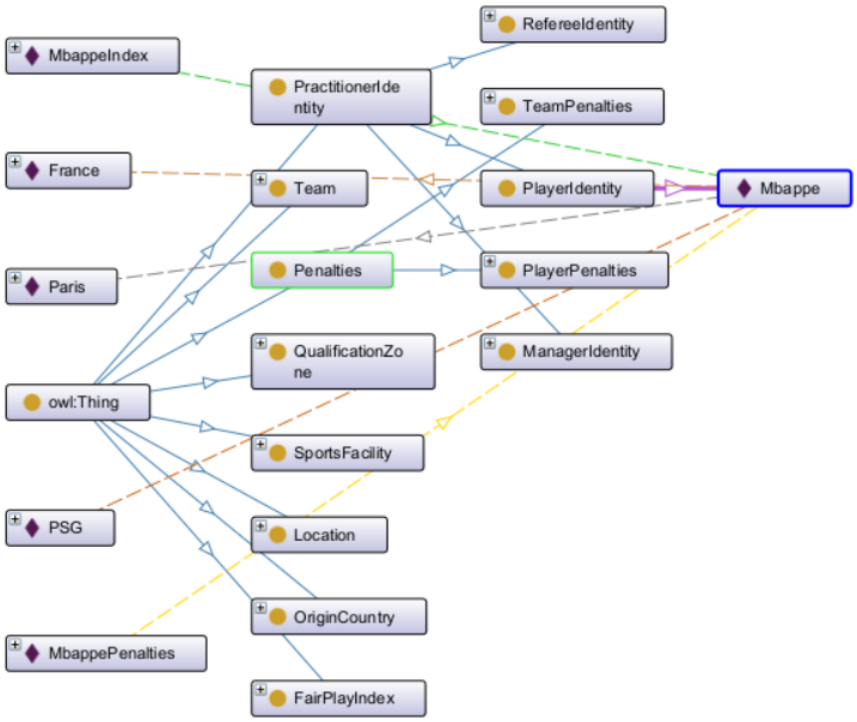

Figure 3: Ontography: Visualizing Interactions in Main Classes with "Protégé".

We can express a minimum and maximum cardinality restriction to specify that a team must have a minimum of 18 players, and also specify that each manager has a minimum of 18 players.

iv. Limitations Encountered with OWL 2 RL in Ontology Development

Despite its utility, OWL 2 RL poses several limitations when applied to ontology construction within our domain. These constraints hinder the precise modeling of concepts and relationships.

Here are some common restrictions of OWL 2 RL in our domain, illustrated with examples:

Limitation of Axiom Expressiveness: OWL 2 RL lacks the capability to express certain types of complex axioms, thus hindering the representation of sophisticated relationships between entities.

Limitation in Modeling Composition Operations: The absence of support for advanced composition operations, such as property intersections, in OWL 2 RL complicates the representation of certain complex relationships. For instance, establishing a relationship that links a player to a team through a management relation by a manager becomes challenging.

Fortunately, despite the constraints posed by OWL 2 RL, these challenges can be addressed through the utilization of SWRL (Semantic Web Rule Language). SWRL provides a means to articulate intricate rules beyond the scope of OWL 2 RL, offering greater flexibility in defining rules and relationships among entities. Additionally, SWRL empowers users to specify customized cardinality rules for entities and relationships, effectively overcoming constraints related to complexity and performance inherent in cardinality restrictions within OWL 2 RL.
Some are illustrated as follows:

- Inferring players who have played for the same team:

  $\text{foot:playsFor}(?p1, ?t) \land \text{foot:playsFor}(?p2, ?t) \rightarrow \text{foot:hasPlayedForSameTeam}(?p1, ?p2)$

- Inferring managers who have managed the same team:

  $\text{foot:manages}(?m1, ?t) \land \text{foot:manages}(?m2, ?t) \rightarrow \text{foot:hasManagedSameTeam}(?m1, ?m2)$

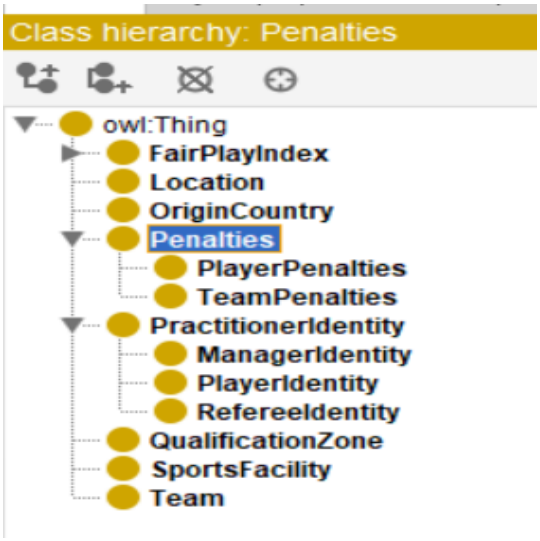

Figure 4: Ontology classes

As detailed in the previous paragraphs, the database normalization improved performance but obscured conceptual clarity. To counter this, we developed an ontology to provide a flexible, semantically-centered schema, facilitating the transformation into RDF triples.

3. R2RML Mapping

i. Semantic Data Mapping: From Relational to RDF

Many initiatives have emerged to aid one in publishing structured resources as Linked Data on the Web with one of the major achievements being the R2RML W3C Recommendation . We present **R2RML-F**, an extension to R2RML, that adopts ECMAScript for capturing domain knowledge and for which we have developed a prototype [2].

The entire ontology is mapped in a fragmented way based on the different classes and object properties obtained in our ontology, the different `.csv` files (more precisely ten files) containing the different data from the tables in our database 1.

Namespace Prefix: `foot: <http://www.foot-info.org/ontology#>`			
Relational database	**RDF**	**Triples Map**	**Notes**
PLAYERS	foot:PlayerIdentity	PlayerIdentityMap	Player data
MANAGERS	foot:ManagerIdentity	ManagerIdentityMap	Manager data
REFEREES	foot:RefereeIdentity	RefereeIdentityMap	Referee data
TEAMS	foot:Team	TeamMap	Team data
COUNTRIES	foot:OriginCountry	OriginCountryMap	Country data
CITIES	foot:Location	LocationMap	City data
CONTINENTS	foot:QualificationZone	QualificationZoneMap	Continent data
PLAYER-STATISTICS	foot:FairPlayIndexPlayer	FairPlayIndexPlayerMap	Player statistics
TEAM-STATISTICS	foot:FairPlayIndexTeam	FairPlayIndexTeamMap	Team statistics
STADIUM	foot:SportsFacility	SportFacilityMap	Stadium
-	foot:PlayerPenalties	PlayerPenaltiesMap	Player penalties
-	foot:TeamPenalties	TeamPenaltiesMap	Team penalties
-	-	PlayerContinentMap	Player - Qualification zone
-	-	RefereeContinentMap	Referee - Qualification zone
-	-	ManagerContinentMap	Manager - Qualification zone
-	-	TeamCountryMap	Team-Country-Qualification

Table 1: Connection between the Relational Database (RDB) entities, the ontology (RDF), and the Triples maps

We choose to use URI Fragments with content negotiation when publishing data for several reasons, considering the specific advantages they offer in different contexts. URI fragments are particularly well-suited for ontologies and vocabularies due to their stability. By assigning fragments to specific concepts or entities within our ontology, we ensure that the identifiers remain stable over time, even if other parts of the URI change. This stability is crucial for maintaining the integrity and coherence of our semantic model, enabling reliable linking and referencing across datasets and applications.

By leveraging both URI fragments and content negotiation, we ensure a comprehensive approach to data publishing that addresses the specific requirements and challenges of different types of data. This approach aligns with the principles of Linked Data, promoting the use of standard web technologies and best practices to maximize the accessibility, interoperability, and usability of our data resources. Our mappings, facilitated by the `mapping.ttl` files and their associated output mapping

`mapping-output.ttl` where the resulting RDF data will be written, adhere to these principles, contributing to a more interconnected and accessible web of data. The base URI used when creating URIs for the resulting RDF resources: `http://www.example.org/foot-infos/resources/`; and the mapping will be achieved by using the .csv files (players.csv, managers.csv, referees.csv, cities.csv, conutries.csv, continents.csv, stdiums.csv, player_statistics.csv and team_statistics.csv) holding the data of our relational database.

We adhere to a naming convention for the triples maps that follows PascalCase, starting with the ontology name and ending with "TriplesMap" to maintain consistency with the conventions taught in our course. As for the IRIs assigned to each generated instance, we adopt an engineering approach where we concatenate our base IRI with the RDF class name and the RDB primary key, employing kebab-case. This approach allows for content negotiation and ensures clarity and coherence in our data representation.

Mapping our relational database to an ontology presented numerous challenges, chiefly regarding data consistency and integrity. One significant hurdle was the complexity involved in performing joins across various tables within the relational database during the ontology development phase. To address this, we required an enriched vocabulary to ensure the preservation of data information.

ii. Mapping strategies

To create a mapping, it's essential to have data instances structured according to the ontology's framework. In our exploration, we identified five scenarios:

(Throughout the examples, we utilize the following prefixes:)

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix foot: <http://www.example.org/foot-infos/ont#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

- Direct mapping from a table (e.g `ManagerIdentityMap`) where all the information is contained in the table :

```
<#ManagerIdentityMap>
    rr:logicalTable [ rr:tableName "MANAGERS" ];
    rr:subjectMap [
        rr:template "managerIdentity#{ID}" ;
        rr:class foot:Manager;
    ] ;
    rr:predicateObjectMap [
        rr:predicate foot:managerFirstName ;
        rr:objectMap [ rr:column "FIRST_NAME"; rr:datatype xsd:
            string ] ;
    ] ;
    rr:predicateObjectMap [
        rr:predicate foot:managerLastName ;
```

```
        rr:objectMap [ rr:column "LAST_NAME"; rr:datatype xsd:
            string ] ;
    ] ...
```

that yields for instance :

```
<http://www.example.org/foot-infos/resources/managerIdentity#4>
    a                       foot:Manager ;
    foot:managerComesFrom   <http://www.example.org/foot-infos/resources/
        country#4> ;
    foot:managerDateOfBirth "1955-07-21"^^xsd:date ;
    foot:managerFirstName   "Marcelo" ;
    foot:managerGenre       true ;
    foot:managerIsBorn      <http://www.example.org/foot-infos/resources/team
        #4> ;
    foot:managerLastName    "Bielsa" ;
    foot:manages            <http://www.example.org/foot-infos/resources/city
        #4> ...
```

- Example of an R2RML mapping snippet of `PlayerPenaltiesMap`:

```
<#PlayerPenaltiesMap>
    rr:logicalTable [ rr:tableName "PLAYER_STATISTICS" ];
    rr:subjectMap [
        rr:template "playerPanalties#{ID}" ;
        rr:class foot:PlayerPenalties;
    ] ;
    rr:predicateObjectMap [
        rr:predicate foot:playerPenaltyDate, rdfs:label ;
        rr:objectMap [ rr:column "STATISTICS_DATE" ; rr:datatype xsd:string ]
            ;
    ];
    rr:predicateObjectMap [
        rr:predicate foot:playerYellowCardNumber, rdfs:label ;
        rr:objectMap [ rr:column "YELLOW_CARDS" ; rr:datatype xsd:integer] ;
    ]; ...
```

that yields for instance:

```
<http://www.example.org/foot-infos/resources/playerIdentity#58>
    a                       foot:Player ;
    rdfs:label              false , "Kim" , "Sophia" ;
    foot:playerComesFrom    <http://www.example.org/foot-infos/resources/
        country#58> ;
    foot:playerDateOfBirth  "1996-04-30"^^xsd:date ;
    foot:playerFirstName    "Sophia" ;
    foot:playerGenre        false ;
    foot:playerIsBornIn     <http://www.example.org/foot-infos/resources/city#
        58> ;
    foot:playerLastName     "Kim" ;
    foot:playsFor           <http://www.example.org/foot-infos/resources/team#
        58> ...
```

- Another R2RML mapping achieved, relies on the presence of the same foreign keys value. For instance the table `Players` and `Stadium` share the same teams foreign key and this enables to

set up a link between the both via a relation `playsAtHome` according to the common valzue of `team_id`.

```
<#PlayerIdentityMap>
    rr:logicalTable [ rr:tableName "PLAYERS" ];
    rr:subjectMap [
        rr:template "playerIdentity#{ID}" ;
        rr:class foot:Player;
    ] ;
    rr:predicateObjectMap [
        rr:predicate foot:playerFirstName, rdfs:label ;
        rr:objectMap [ rr:column "FIRST_NAME" ; rr:datatype xsd:string] ;
    ] ;...
   rr:predicateObjectMap [
        rr:predicate foot:playsAtHome;
        rr:objectMap [
            rr:parentTriplesMap <#SportFacilityMap>;
            rr:joinCondition [
                rr:child "TEAM_ID"; rr:parent "TEAM_ID";
            ];
        ];
    ] .
```

That yields for instance:

```
    a                       foot:PlayerIdentity , foot:Player ;
        rdfs:label              "Jr." , true , "Neymar" ;
        foot:playerAttendsIn    <http://www.example.org/foot-infos/resources/
            qualificationZone#3> ;
        foot:playerComesFrom    <http://www.example.org/foot-infos/resources/
            country#45> ;
        foot:playerDateOfBirth  "1992-02-05"^^xsd:date ;
        foot:playerFirstName    "Neymar" ;
        foot:playerGenre        true ;
        foot:playerIsBornIn     <http://www.example.org/foot-infos/resources/
            city#45> ;
        foot:playerLastName     "Jr." ;
        foot:playsAtHome        <http://www.example.org/foot-infos/resources/
            stadium#45> ;
        foot:playsFor           <http://www.example.org/foot-infos/resources/
            team#45> .
...
```

- Mapping from joined tables (e.g `TeamCoountryMap`) where the information is contained across several tables.

```
<#TeamCountryView>
    rr:logicalTable [
        rr:sqlQuery """
SELECT␣teams.id,␣teams.name␣AS␣tname,␣countries.name␣AS␣country_name,␣
    continents.name␣AS␣continent_name
FROM␣teams
LEFT␣OUTER␣JOIN␣cities␣ON␣teams.city_id␣=␣cities.id
LEFT␣OUTER␣JOIN␣countries␣ON␣cities.country_id␣=␣countries.id
LEFT␣OUTER␣JOIN␣continents␣ON␣countries.continent_id␣=␣continents.id
```

```
␣␣␣␣␣␣␣␣"""";
    ];
    rr:subjectMap [
        rr:template "teams#{ID}" ;
        rr:class foot:Team;
    ] ;
    rr:predicateObjectMap [
        rr:predicate foot:teamName, rdfs:label ;
        rr:objectMap [ rr:column "TNAME"; rr:datatype xsd:string ] ;
    ] ;
  ....
```

That yields for instance :

```
<http://www.example.org/foot-infos/resources/country#38>
    a                           foot:Country ;
    rdfs:label                  "Sweden" ;
    foot:countryConsistsOf      <http://www.example.org/foot-infos/resources/
        city#38> ;
    foot:countryName            "Sweden" ;
    foot:isIncludedIn           <http://www.example.org/foot-infos/resources/
        qualificationZone#1> ;
    foot:playerNativeCountry    <http://www.example.org/foot-infos/resources/
        playerIdentity#67> ;
    foot:refereeNativeCountry   <http://www.example.org/foot-infos/resources/
        refereeIdentity#58> .
...
```

Indeed, the query provides detailed information on the teams, including their names, as well as the names of the associated countries and continents, even though some of these relationships may not exist (thanks to **LEFT OUTER JOIN**). This type of mapping is very useful for integrating relational data into the Semantic Web, enabling this data to be represented and linked in a standardised and interoperable format.

- We can also deduce, using the foreign keys present in the ‘players‘ and ‘stadiums‘ tables, that a player plays at home:

```
<#TeamCountryView>
    rr:logicalTable [
        rr:sqlQuery """
SELECT␣teams.id,␣teams.name␣AS␣tname,␣countries.name␣AS␣country_name,␣
    continents.name␣AS␣continent_name
FROM␣teams
LEFT␣OUTER␣JOIN␣cities␣ON␣teams.city_id␣=␣cities.id
LEFT␣OUTER␣JOIN␣countries␣ON␣cities.country_id␣=␣countries.id
LEFT␣OUTER␣JOIN␣continents␣ON␣countries.continent_id␣=␣continents.id
␣␣␣␣␣␣␣␣"""";
    ];
    rr:subjectMap [
        rr:template "teams#{ID}" ;
        rr:class foot:Team;
    ] ;
    rr:predicateObjectMap [
        rr:predicate foot:teamName, rdfs:label ;
        rr:objectMap [ rr:column "TNAME"; rr:datatype xsd:string ] ;
```

```
    ] ;
  ....
```

The output is as follows:

```
<http://www.example.org/foot-infos/resources/playerIdentity#63>
    a                       foot:PlayerIdentity , foot:Player ;
    rdfs:label              "Junior" , "Vinicius" , true ;
    foot:isCoachedBy        <http://www.example.org/foot-infos/resources/
        managerIdentity#31> , <http://www.example.org/foot-infos/resources/
        managerIdentity#74> ;
    foot:playerAttendsIn    <http://www.example.org/foot-infos/resources/
        qualificationZone#4> ;
    foot:playerIsBornIn     <http://www.example.org/foot-infos/resources/city#
        34> ;
    foot:playerLastName     "Junior" ;
    foot:playsAtHome        <http://www.example.org/foot-infos/resources/
        stadium#34> ;
    foot:playsFor           <http://www.example.org/foot-infos/resources/team#
        34> .
```

The mapping approach described above demonstrates how relational data can be effectively integrated into the Semantic Web, allowing for standardized and interoperable representation of this data. However, it is crucial to ensure that the data remains consistent and reliable throughout this process. To achieve this, we can employ SPARQL queries to validate and verify the integrity of our relational database.

4. Ensuring Relational Database Integrity via SPARQL Queries

The mapping approach described above demonstrates how relational data can be effectively integrated into the Semantic Web, allowing for standardized and interoperable representation of this data. However, the complexity of the queries needed to retrieve meaningful insights can be significant. For instance, consider the following SPARQL query:

```
PREFIX foot: <http://www.example.org/foot-infos/ont#>
SELECT ?managerFirstName ?managerLastName ?name WHERE {
  ?manager foot:managerAttendsIn ?zone.
  ?manager a foot:ManagerIdentity;
         foot:managerLastName ?managerLastName;
         foot:managerFirstName ?managerFirstName.
  ?zone a foot:QualificationZone;
           foot:qualificationZoneName ?name.
}
```

Listing 3: SPARQL query to retrieve manager information and their qualification zone

This SPARQL query is straightforward and easy to understand, leveraging the flexibility of RDF to traverse relationships effortlessly. Now, consider the equivalent SQL query:

```
SELECT managers.first_name, managers.last_name, qualification_zones.name AS
    zone_name
FROM managers
JOIN manager_qualification_zones ON managers.id = manager_qualification_zones.
    manager_id
```

```
JOIN qualification_zones ON manager_qualification_zones.zone_id =
    qualification_zones.id
ORDER BY managers.last_name, managers.first_name;
```

Listing 4: SQL query to retrieve manager information and their qualification zone

The SQL query requires multiple joins and explicit table references, making it more complex and less flexible. The use of SPARQL simplifies the process, especially when dealing with extensive and interconnected data sets. Transforming relational databases into RDF and utilizing SPARQL not only enhances query simplicity but also promotes better data integration and interoperability.

SPARQL

- Simplicity Syntax:
  - SPARQL syntax is very intuitive when traversing RDF graphs.
  - Relationships between entities are naturally expressed using triples.
- Flexibility:
  - With SPARQL, data can be queried flexibly without worrying about the rigid constraints of relational table structures.
  - SPARQL allows easy extension of queries to include new relationships or properties without major modifications.
- Interoperability:
  - Using RDF ontologies provides better data interoperability across different systems and domains.
  - Standardized ontologies facilitate understanding and integration of data.

SQL

- Simplicity Syntax:
  - Multiple joins can make SQL queries complex, especially when relationships between entities are not immediately obvious.
- Flexibility:
  - Adding new relationships or properties in an SQL query can require significant changes to the query structure, notably by adding more joins.
- Interoperability:
  - Although SQL is powerful for relational databases, it lacks standards for interoperability beyond specific relational systems.

5. Conclusion

The complexity of SQL queries highlights the challenges of extracting comprehensive information from relational databases. To overcome these obstacles and enhance data interoperability, transforming our relational database into RDF via ontology establishment and R2RML mappings is advantageous. This enables more flexible and powerful querying through SPARQL and better integration with other data sources on the Semantic Web. However, despite the rich semantics and improved interoperability offered by this process, the absence of linked data deployment in the front-end limits potential benefits. End users cannot fully exploit the flexibility and richness of the data without interactive and intuitive interfaces. Deployment of the ontology using tools like Widoco and the data using tools such as Apache Jena, brwsr, and Docker is crucial for facilitating this process. It is essential to develop suitable front-end applications that allow intuitive navigation and querying of RDF data. Continuous validation of data integrity through SPARQL queries is also necessary. In conclusion, while transforming relational data into RDF presents challenges, it paves the way for increased interoperability and an enriched user experience, provided these data are effectively deployed in the front-end.

Bibliography

[1] Guntars Bumans. Mapping between relational databases and owl ontologies: an example. Scientific Papers, University of Latvia, 756:99–117, 2010.

[2] Christophe Debruyne and Declan O'Sullivan. R2rml-f: Towards sharing and executing domain logic in r2rml mappings. LDOW@ WWW, 1593, 2016.