World Cup Soccer Database

Rebecca Thomas, Dmytry Berkout

April 14, 2016

Contents

1 Purpose **3**
1.1 Purpose of the Document . . . 3
1.2 Purpose of the Project . . . 3
1.3 Purpose of Phase 1 . . . 3

2 Problems and Solutions **3**

3 Assumptions **4**

I Phase 1 Documentation **4**

4 Environment and Requirements Analysis **4**
4.1 Using MondialDB . . . 4
4.2 Extract Transform Load Tool . . . 5
4.3 Top-Level Information Flow Diagram . . . 5

5 List of Tasks and Task Flow Diagram **7**
5.1 Extract, Transform, and Load Task . . . 8
5.2 Web Query Processor . . . 10
5.3 Webpage SQL Query Processor Task . . . 11
5.4 Output Results Task . . . 12

6 List of Documents **13**
6.1 Queries . . . 14

II Phase 2 Documentation **14**

7 ER Model **15**

8 Relational Schema **15**

9 Query for Super Stars **16**

10 Query for a Given Player 17

11 Query for Team History 18

12 Query for a given Cup 18

13 Country's Players Query 19

14 Pseudo code for Extract and Transform Task 19

15 Pseudo code For Load Task 22

1 Purpose

1.1 Purpose of the Document

This document is an introduction to the MondialDB project. It will provide details on the implementation and purpose of MondialDB. This document will provide a detailed design of MondialDB. It will include an information flow diagram and task forms that go into the inner workings of the project. It will also include the problems we encountered and the solutions we had for them.

1.2 Purpose of the Project

The purpose of the project is to create a database that can access various details about the World Cup. It will contain soccer team names and members and statistics such as player records, team records, penalty information, and the rank obtained. Another major part of this project is creating a tool to extract information about the World Cup from the Internet. It will be able to extract information from various websites and put them into a database readable format. The tool will be able to deal with data from European and American websites and transform it into a standard format. This project will allow for easy and readable queries.

1.3 Purpose of Phase 1

In this phase of the project we are to design the database and receive feedback on it. We will attempt to catch any significant errors in design at this stage and prevent a broken or severely flawed implementation from going through. If design errors are caught early it will require considerably less work to fix them.

2 Problems and Solutions

The following list is composed of problems we encountered with the conceptual design of the project.

- Problem: We lack database design experience. Without experience, it is difficult to design a database.

  Solution: Follow project examples and study how databases are designed.

- Problem: We don't understand the World Cup and we don't know where to find data for it.

  Solution: Find out how the World Cup works. Research various World Cup websites. Find several that are usable for our project.

- Problem: We don't know how to extract data from websites.

  Solution: Research data extraction. Make a very basic test script. See what suites are available for use.

- Problem: We don't know how to create websites.

  Solution: Research the creation of websites. Make a very basic website. See what suites are available for use.

- Problem: We don't live close together.

  Solution: Use text messages, phone calls, and email to communicate. Use Google documents to share data.

3 Assumptions

The assumptions we made are the following:

- Our web server will not be overloaded despite not having restrictions on who can use it.
- Our web server software will not fail.
- The database software will be sufficient for the scope of this project.
- The data pulled from websites is accurate.
- We will not need multilingual support. We will support only English.
- We will not need handicap support for our website.
- We will not need a mobile accessible version for our website.
- Our users are average people who do not have a computer science background but are able to comfortably use the internet.

Part I
Phase 1 Documentation

4 Environment and Requirements Analysis

4.1 Using MondialDB

The user will interact with MondialDB through our website. The user will connect to the website using a web browser and a simple webpage will be displayed. The sole purpose of this website is to run specific queries from the web-server through MondialDB and send the results back to the user. The website will only contain a selection of the predefined queries. On selection, a website form will appear which will contain the necessary information to perform the specified query. Both the website and web-server will check the input for validity. Once the query is processed, a table of results will appear underneath the form.

4.2 Extract Transform Load Tool

For this project we will write a python script to pull data from selected websites. The ideal script will be as simple as possible while robust enough to work with a number of different websites. We will input a list of websites and the tool should automatically convert the data into database format and insert it, or provide a script to insert it into MondialDB.

4.3 Top-Level Information Flow Diagram

See Figure 1 on page 6 for the Top-Level Information Flow Diagram. The flow is generally as follows:

1. Data is input into MondialDB from Soccer Websites through Extract-Transform-Load.
2. The user asks for the website and is provided it through the Webpage Server.
3. The Web Query Processor decides what kind of query the user is asking for.
4. The Webpage SQL Query Processor translates the user query into SQL to be executed on MondialDB.
5. The query is executed and database results are returned from MondialDB
6. The results are outputted to the user with the format depending on the kind of query.

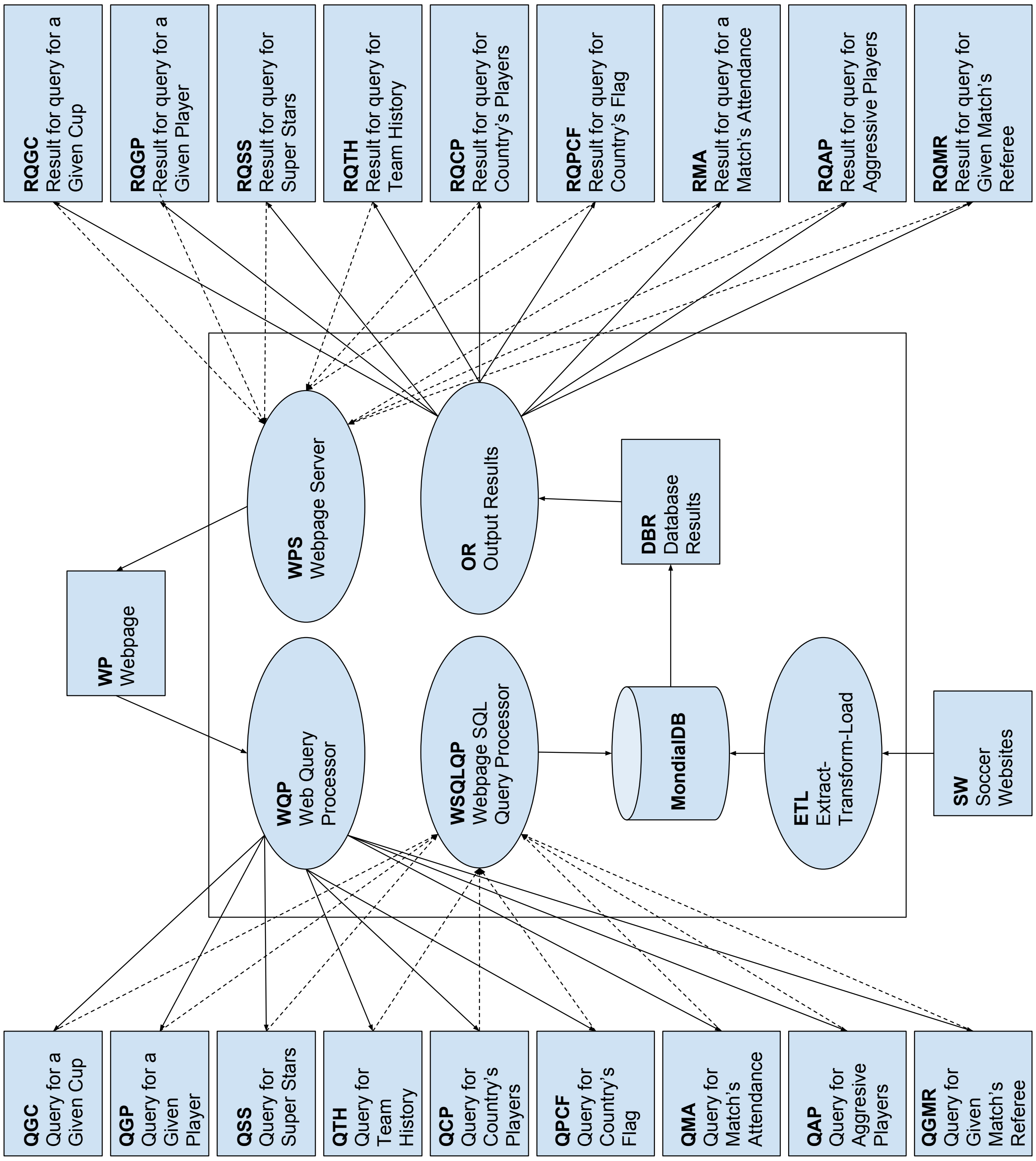
RQGC Result for query for a Given Cup
RQGP Result for query for a Given Player
RQSS Result for query for Super Stars
RQTH Result for query for Team History
RQCP Result for query for Country's Players
RQPCF Result for query for Country's Flag
RMA Result for query for a Match's Attendance
RQAP Result for query for Aggressive Players
RQMR Result for query for Given Match's Referee
WPS Webpage Server
OR Output Results
DBR Database Results
WP Webpage
WQP Web Query Processor
WSQLQP Webpage SQL Query Processor
MondialDB
ETL Extract-Transform-Load
SW Soccer Websites
QGC Query for a Given Cup
QGP Query for a Given Player
QSS Query for Super Stars
QTH Query for Team History
QCP Query for Country's Players
QPCF Query for Country's Flag
QMA Query for Match's Attendance
QAP Query for Aggresive Players
QGMR Query for Given Match's Referee

Figure 1: Information Flow Diagram

5 List of Tasks and Task Flow Diagram

We describe the tasks and subtasks necessary to make, populate, and query MondialDB. See Figure 2 on page 7 for the Task Flow Diagram. The tasks are the following:

- Extract, Transform, and Load Task
- Webpage Server Task
- Web Query Processor
- Webpage SQL Query Processor
- Output Results Task

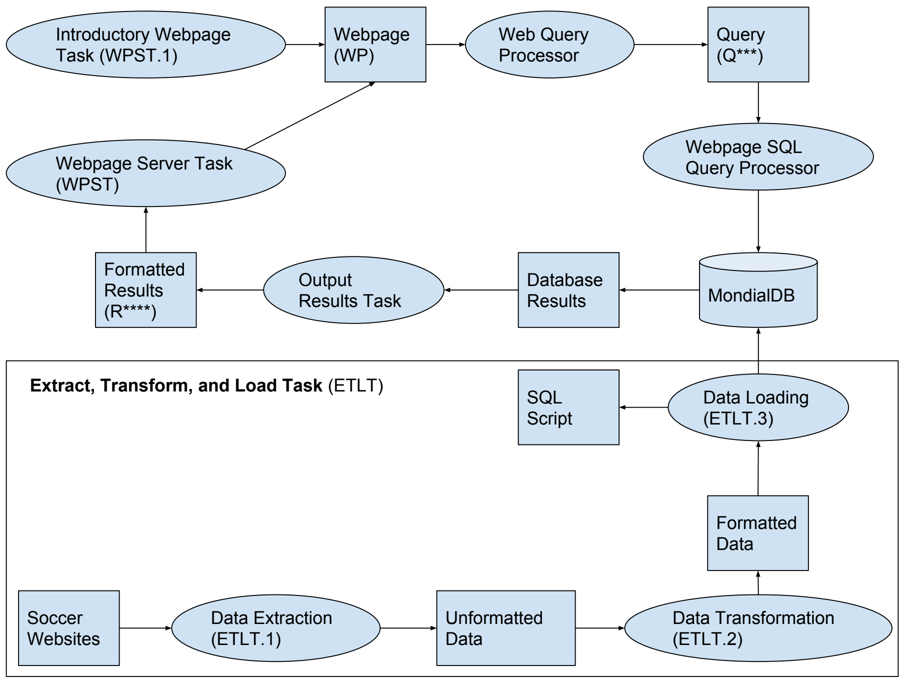

Figure 2: Task Flow Diagram

5.1 Extract, Transform, and Load Task

Task Label	ETLT
Task Name	Extract, Transform, and Load Task
Performer	Python script
Purpose	To extract data from Soccer Websites, transform it into a usable format, and send it to MondialDB
Enabling Condition	On database creation or database update
Description	It takes the information from Soccer Websites and puts the information into MondialDB.
Frequency	On database update
Duration	It will depend on the extraction, transformation, and load subtasks.
Importance	Most important
Maximum Delay	It depends on the subtasks.
Input	Soccer Websites
Output	Copy of the SQL script run by the python script as well as debugging information
Document Use	Soccer Websites, Unformatted Data, Formatted Data
Operations Performed	Data extraction, data transformation, and data loading
Subtasks	Data extraction (ETLT.1), data transformation (ETLT.2), data loading (ETLT.3)
Error Conditions	Errors from subtasks

5.1.1 Data Extraction

Task Label	ETLT.1
Task Name	Data Extraction
Performer	Python script
Purpose	To extract data from Soccer Websites
Enabling Condition	On database data insertion
Description	It pulls HTML from the Soccer Websites. It then parses the HTML for data to put into MondialDB.
Frequency	On database update
Duration	It depends on how quickly websites are scraped and extracted.
Importance	Most important
Maximum Delay	It depends on how many Soccer Websites are chosen.
Input	Soccer Websites
Output	Unformatted Data from the Soccer Websites
Document Use	Soccer Websites -> Unformatted Data
Operations Performed	Data extraction
Subtasks	None
Error Conditions	Soccer Websites are invalid. The format of the website is invalid or confusing.

5.1.2 Data Transformation

Task Label	ETLT.2
Task Name	Data Transformation
Performer	Python script
Purpose	To transform data from Soccer Websites into a standardized format
Enabling Condition	On database data insertion
Description	It standardizes the data produced by Data Extraction. For example, names that are formatted like "Last Name, First Name" and "First Name Last Name" shall be changed into a standard format.
Frequency	On database update
Duration	It depends on how quickly the data goes from being unformatted to being formatted.
Importance	Important
Maximum Delay	It depends on how badly the original data was formatted.
Input	Unformatted Data from Data Extraction
Output	Formatted Data
Document Use	Unformatted Data -> Formatted Data
Operations Performed	Data transformation
Subtasks	None
Error Conditions	The data are formatted badly.

5.1.3 Data Loading

Task Label	ETLT.3
Task Name	Data Loading
Performer	Python script
Purpose	To load formatted data into MondialDB
Enabling Condition	On database data insertion
Description	Makes an SQL script that inserts the standardized data from Data Transformation into MondialDB.
Frequency	On database update
Duration	It depends on how quickly the data is inserted into MondialDB
Importance	Most important
Maximum Delay	It depends on how the slow the connection between the data collector and the database is.
Input	Formatted Data
Output	Log from inserting into MondialDB. Copy of the SQL script run by the python script.
Document Use	Formatted Data -> SQL Script
Operations Performed	Data Loading
Subtasks	None
Error Conditions	There is a faulty connection with the database.

5.2 Web Query Processor

Task Label	WQPT
Task Name	Web Query Processor
Performer	Web-server
Purpose	Processes queries from the web-server and sends the respective query to the Webpage SQL Query Processor Task.
Enabling Condition	After web-server runs
Description	It determines the user-specified query from the forms on the Webpage. Once this query is validated, it sends the query on to the Webpage SQL Query Processor.
Frequency	Always on
Duration	As long as the database is active
Importance	Important
Maximum Delay	Response delay to user requests should be short and less than 10 seconds.
Input	User-inputted form data from the Webpage
Output	Sends Query type to Webpage SQL Processor Task. See Query Types.
Document Use	Webpage -> Query
Operations Performed	Validates queries and determines query type.
Subtasks	None
Error Conditions	The query isn't valid.

5.3 Webpage SQL Query Processor Task

Task Label	WSQLQPT
Task Name	Webpage SQL Query Processor Task
Performer	Webpage server
Purpose	Sends the query from the user-specified query task to MondialDB
Enabling Condition	User submits valid query
Description	The WSQLQPT is the final step before the query reaches MondialDB. It will send a perform a raw SQL query in MondialDB
Frequency	Triggers on every received valid user query
Duration	The SQL shouldn't take longer than 15 seconds to run.
Importance	Important
Maximum Delay	The SQL shouldn't take at most longer than 45 seconds to run.
Input	User-specified query from the following list: QGC (Query for a given cup) QGP (Query for a given player) QSS(Query for Super Stars) QTH(Query for Team History) QCP(Query for Country's Players) QPCF(Query for Country's Flag) QMA (Query for Match's Attendance) QAP (Query for Aggressive Players) QGMR (Query for a Given Match's Referee)
Output	Sends SQL query to MondialDB
Document Use	Query
Operations Performed	If the query is a valid query from the aforementioned list, then create the SQL command. If it is not a valid query, send an error message back.
Subtasks	None
Error Conditions	The user-specified query is not valid.

5.4 Output Results Task

Task Label	OR
Task Name	Output Results Task
Performer	Webpage Server
Purpose	Sends the query results to the user
Enabling Condition	MondialDB receives query and sends output to Output Results Task
Description	The WSQLQPT is the final step before the query reaches MondialDB. It will send a raw SQL query to MondialDB.
Frequency	Triggers on every received request
Duration	In ideal network conditions, it should be short and less than 10 seconds.
Importance	Most Important
Maximum Delay	Response delay to user requests should be short and less than 10 seconds.
Input	For a user-specified query, Database Results
Output	The Formatted Result from the results
Document Use	Database Results -> Formatted Results
Operations Performed	Sends query results to Webpage Server Task
Subtasks	None
Error Conditions	MondialDB gave back error conditions instead of valid results. Connection to user is disrupted. Web server becomes overloaded.

6 List of Documents

See page 13 for repeated Task Flow Diagram. The documents are the following:

- Webpage
- Query
- Database Results
- Formatted Results
- Soccer Websites
- Unformatted Data
- Formatted Data
- SQL Script

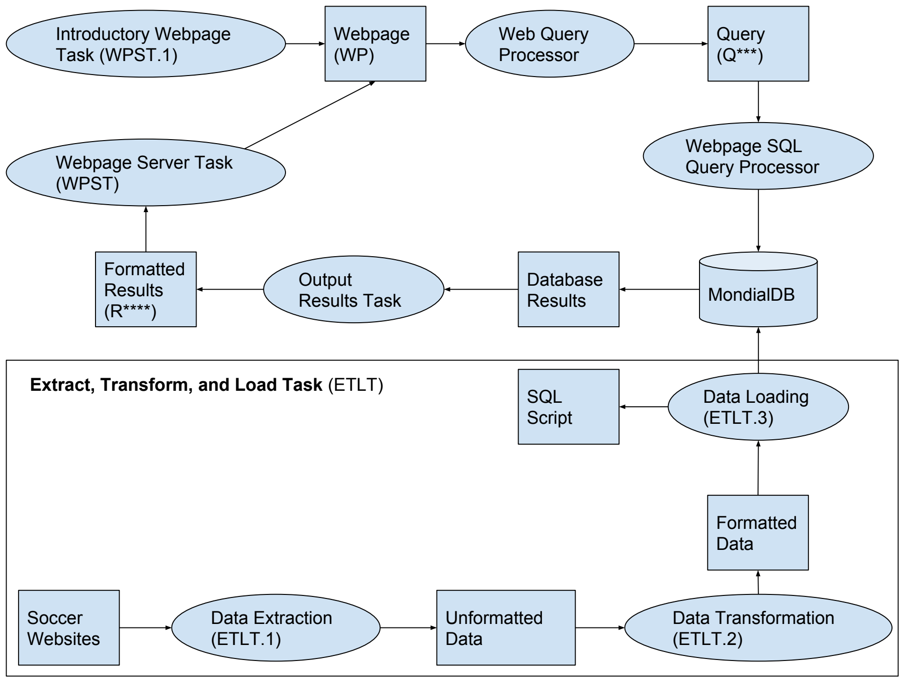

Figure 3: Task Flow Diagram

6.1 Queries

The types of Queries:

- QGC (Query for a given cup)
- QGP (Query for a given player)
- QSS (Query for Super Stars)
- QTH (Query for Team History)
- QCP (Query for Country's Players)
- QPCF(Query for Country's Flag)
- QMA (Query for Match's Attendance)
- QGMS (Query for Aggressive Players)
- QGMR (Query for a Given Match's Referee)

Query for Country's Flag This query returns information about a country's flag. Most notably this includes a link to a picture of that country's flag. The query is interesting because it adds an extra dimension to the database. Users might find it easier to recognize a country by its flag instead of by its name. Others might be curious and be inspired to learn more interesting information about flags. Flags traditionally represent countries. They are symbols for the people of that country to rally around and to represent. They represent the history of the nation. Thus, they are interesting.

Query for Match's Attendance This query returns information about a specific match. It will give back what stadium the match was held in. It will also tell the user how many people attended that particular match. Attendance information is wonderfully informative. The user could try to find out what teams are most popular, what stadiums are most popular, or what host countries are most popular.

Query for Aggressive Players This query tells users who the most aggressive players are. The metric of aggressiveness is determined by how many penalties they accumulate. It is interesting to see the human side of soccer and how vicious it can get.

Query for a Given Match's Referee This query tells users what referee was calling for a particular match. Many people assume referees are biased against the team that they like. By using this query, they could see if the referee really is biased against their favorite team.

Part II

Phase 2 Documentation

7 ER Model

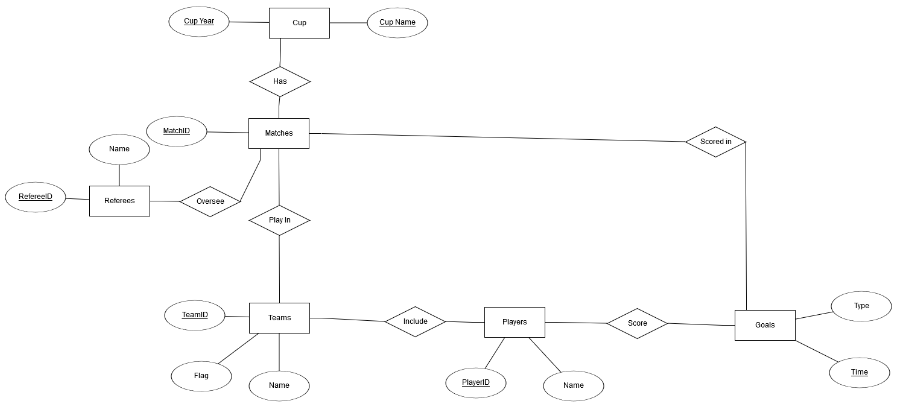

Figure 4: ER Model

8 Relational Schema

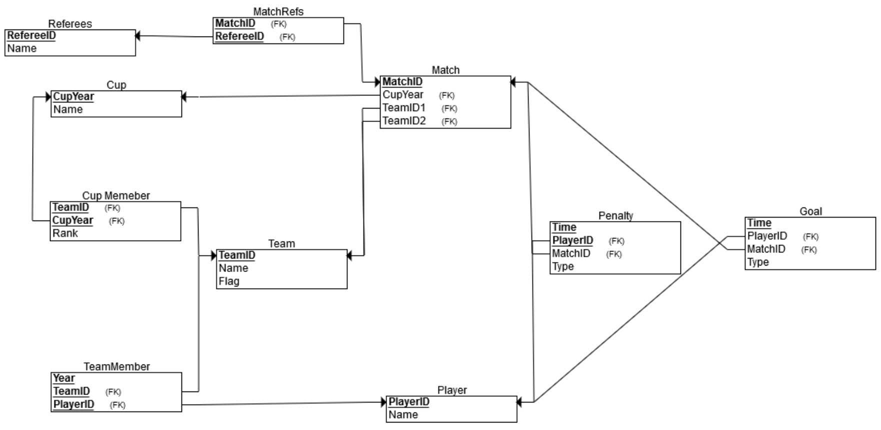

Figure 5: Relational Schema

9 Query for Super Stars

```
--Get super stars
CREATE VIEW super_stars AS
SELECT tm1.playerID ,cm.CupYear
FROM CupMember cm, TeamMember tm1, TeamMember tm2.
WHERE tm1.playerID==tm2.playerID
AND tm1.year!=tm2.year
AND cm.TeamID=tm1.teamID
```

10 Query for a Given Player

```
--Players with said name
CREATE VIEW player_ids AS
SELECT p.playerID
FROM player p
WHERE p.name=selected_name

--Teams those players were on
CREATE VIEW team_ids AS
SELECT tm.teamID, pids.playerID
FROM TeamMember tm,player_ids pids
WHERE tm.playerID=pids.p

--Team Names
SELECT t.name
FROM team_ids tids, team t
WHERE t.teamID=tids.teamID

--Cups those players were in
CREATE VIEW cup_years AS
SELECT cm.CupYear, tid.PlayerID
FROM CupMember cm, team_ids tids
WHERE cm.teamID=                tids.TeamID

--Regular Goals
SELECT g.playerID, count(g.playerID)
FROM Goal g, player_ids pids
WHERE g.type= Regular  and g.PlayerID==pid.playerId
GROUP BY g.playerID

--Penalty shots
SELECT g.playerID, count(g.playerID)
FROM Goal g, player_ids pids
WHERE g.type= Penalty  and g.PlayerID==pid.playerId
GROUP BY g.playerID
```

11 Query for Team History

```
SELECT c.name, c.year, t1.name, t2.name
FROM  cup c. team t2,match m
WHERE m.cupyear=c.cupyear
AND m.teamID1=t1.teamID
AND m.teamID2=t2.teamID

SELECT c.Name,t.name , cm.cupyear, cm.rank
FROM team t, cupmember cm, cup c
WHERE t.teamId=cm.teamId
AND c.cupyear=cm.cupyear

SELECT p.name,m.name,p.time,p.type
FROM penalty p, match m,player pl
WHERE m.matchID=p.matchID
AND p1.playerID=p.playerID
```

12 Query for a given Cup

```
--Teams
CREATE VIEW participating_team_ids AS
SELECT cm.teamID
FROM CupMember cm
WHERE cm.CupYear=SelectedYear;

--Ranks
SELECT t.name, cm.rank
FROM participating_team_ids ids, CupMember cm,Team t
WHERE ids.teamID==t.TeamID
AND ids.teamID==cm.teamID
AND cm.CupYear=selected year

--Regular Goals
SELECT g.matchID, count(g.time)
FROM Goal g, match m
WHERE g.type= Regular  and g.matchID==m.matchID
GROUP BY g.matchID

--Penalty shots
SELECT g.matchID, count(g.time)
FROM Goal g, match m
WHERE g.type= Penalty  and g.matchID==m.matchID
GROUP BY g.matchID
```

13 Country's Players Query

```
--list of teams in cups
CREATE VIEW teams_ids AS
SELECT cm.teamIds
FROM CupMember cm
--Matches teams participated in
CREATE VIEW matchs_ids AS
 Distinct(SELECT m.teamID1,m.matchID
FROM match m
JOIN
SELECT m.teamID2, m.matchID
FROM match m )
```

14 Pseudo code for Extract and Transform Task

The pseudo code for extracting can be thoroughly extended. The basic process is to grab the html of a website, parse it, and then execute a regex on its contents to capture data. The following is a code snippet:

```
import requests
import re
import pprint
from bs4 import BeautifulSoup

debug = True

def log(my_string):
    "Ensure that errors are propagated to the user"
    print my_string

def debug_print(my_string):
    "When debugging, print this out"
    if debug:
        print my_string

def pretty_print_dict(my_dict):
    "Pretty print the dictionary"
    pp = pprint.PrettyPrinter(depth=3)
    pp.pprint(my_dict)

def get_teams(website):
    "Get the links of the teams"
    #Get the website html
    page = requests.get(website)
    #Parse the website html
    soup = BeautifulSoup(page.content, 'html.parser')
    #Make a regex that will find the links that go to team-specific pages
    regex = re.compile('team')
    #Execute that regex (look for links matching the regex)
    all_teams_html = soup.find_all(href=regex)
    #Initialize the team dictionary
    team_dictionary = {}
    #For each team, look at its html and extract information
    for html in all_teams_html:
        #Acquire the name of the country from its html
        possible_names = html.find_all(class_='team-name')
        if possible_names != []:
            #Use the first name
            country_name = possible_names[0].get_text()
        else:
            #No names were found
            country_name = "INVALIDCOUNTRY"
            log("WARNING:␣In␣finding␣teams,␣we␣could␣not␣establish␣a␣team␣name.␣

        #Add the team's webpage to the team dictionary under link
        team_dictionary[country_name]= {'link' : (html['href'])}
    #Return the team dictionary
    return team_dictionary

def get_team_data(base, team):
    "For␣the␣given␣team,␣acquire␣general␣world␣cup␣data"
    #Find the team's webpage
    link = team["link"]
    #Get the website html
    page = requests.get(base + link)
    #Parse the website html
    soup = BeautifulSoup(page.content, 'html.parser')
    #Find all statistical data and appropriate name
    stats_names = soup.find_all(class_="label-name")
    team_stats = soup.find_all(class_="label-data")
    #For every stat, put the stats in this team's part of the dictionary
    for stat_name_html, value_html in zip(stats_names, team_stats):
        stat_name = stat_name_html.get_text()
        value = value_html.get_text()
        debug_print(stat_name)
        debug_print(value)
        if stat_name != "␣":
            #add value to the list of values at team[stat_name]
            team.setdefault(stat_name, []).append(value)
    debug_print("————————————")
```

```
def get_all_teams_data(base, teams_dict):
    "For␣the␣given␣teams,␣acquire␣general␣world␣cup␣data␣for␣each"
    for team in teams_dict:
        get_team_data(base,teams_dict[team])

#Main section, do this:
base = "http://www.fifa.com"
teams_website = base + '/fifa-tournaments/teams/search.html'
team_dictionary = get_teams(teams_website)
get_all_teams_data(base,team_dictionary)
```

15 Pseudo code For Load Task

The following is a code snippet:

```
import MySQLdb

debug = True


# Open database connection
db = MySQLdb.connect("localhost","user","password","MondialDB" )
cursor = db.cursor()

#Drop the previous tables
drop_existing_tables(cursor)
#Load in the new data

#Disconnect from server
db.close()

def safe_execute(cursor, sql):
    try:
        # Execute the SQL command
        cursor.execute(sql)
        # Commit changes in the database
        db.commit()
    except:
        # Rollback in case there is any error
        db.rollback()

def drop_existing_tables(cursor):
    "Drop certain tables if they already exist"
    safe_execute(cursor,"DROP TABLE IF EXISTS CUP")
    safe_execute(cursor,"DROP TABLE IF EXISTS CUP_MEMBER")
    safe_execute(cursor,"DROP TABLE IF EXISTS TEAM_MEMBER")
    safe_execute(cursor,"DROP TABLE IF EXISTS TEAM")
    safe_execute(cursor,"DROP TABLE IF EXISTS PLAYER")
    safe_execute(cursor,"DROP TABLE IF EXISTS PENALTY")
    safe_execute(cursor,"DROP TABLE IF EXISTS GOAL")
    safe_execute(cursor,"DROP TABLE IF EXISTS MATCH")

def create_player_table(cursor):
    sql =
    """CREATE TABLE PLAYER (
    FIRST_NAME  CHAR(20) NOT NULL,
    LAST_NAME  CHAR(20)),
    PID INTEGER,
    PRIMARY KEY (PID)
    """
    safe_execute(cursor, sql)
```