Database project CS-322

Release 0.2

Amos Wenger, Sebastien Zurfluh & Yoan Blanc

April 29, 2012

CONTENTS

CHAPTER

ONE

DELIVERABLE 1

The goal of this deliverable is to design an ER model, a corresponding relational schema and create the database tables in the given database. The organization of the data in files and the given description does not imply neither an ER model nor a relational schema. It is given to help the student understand the format of the data faster. Finally, a discussion about constraints and removing redundant information is expected.

1.1 Create the ER model for the data

1.1.1 The Schema

See the figure on next page.

1.1.2 Decision made building the Entity-Relationship schema

We tried to remove every denormalized fields, meaning fields that are representing information you can obtain using the rest of the data set.

People

- *Players* and *Coaches* are pointing to the same entities the coach's *coach_id* and the player's *ilkid* are defining the same thing (e.g. *MOEDO01*). So we grouped them under the *Person* entity.
- *Player*'s height will be converted into inches, *1ft = 12in*.
- *Player*'s fields like *first_season*, *last_season* or *college* can be obtained from the *Draft* and *Player Season* entities. They are just denormalized fields.

Teams

The teams pretty look like the CSV file, except that we removed the league from it since some teams switched from *ABA* to *NBA* when *ABA* merged into the later.

Conference and leagues

According to Wikipedia, a team belongs to a *Division* which belongs to a *Conference* (being *Eastern* and *Western*). We first linked the team to a *Conference* but the dataset gives the information about the *Conference* only on *All Star* games (where the best players of each *Conferences* create an *All Star* team and play against each other). So, the *Conference* information will only live there because of the dataset.

About *Leagues*, there are two of them *ABA* and *NBA* and a team may have changed during the year 1975 when *ABA* got merged into the *NBA*.

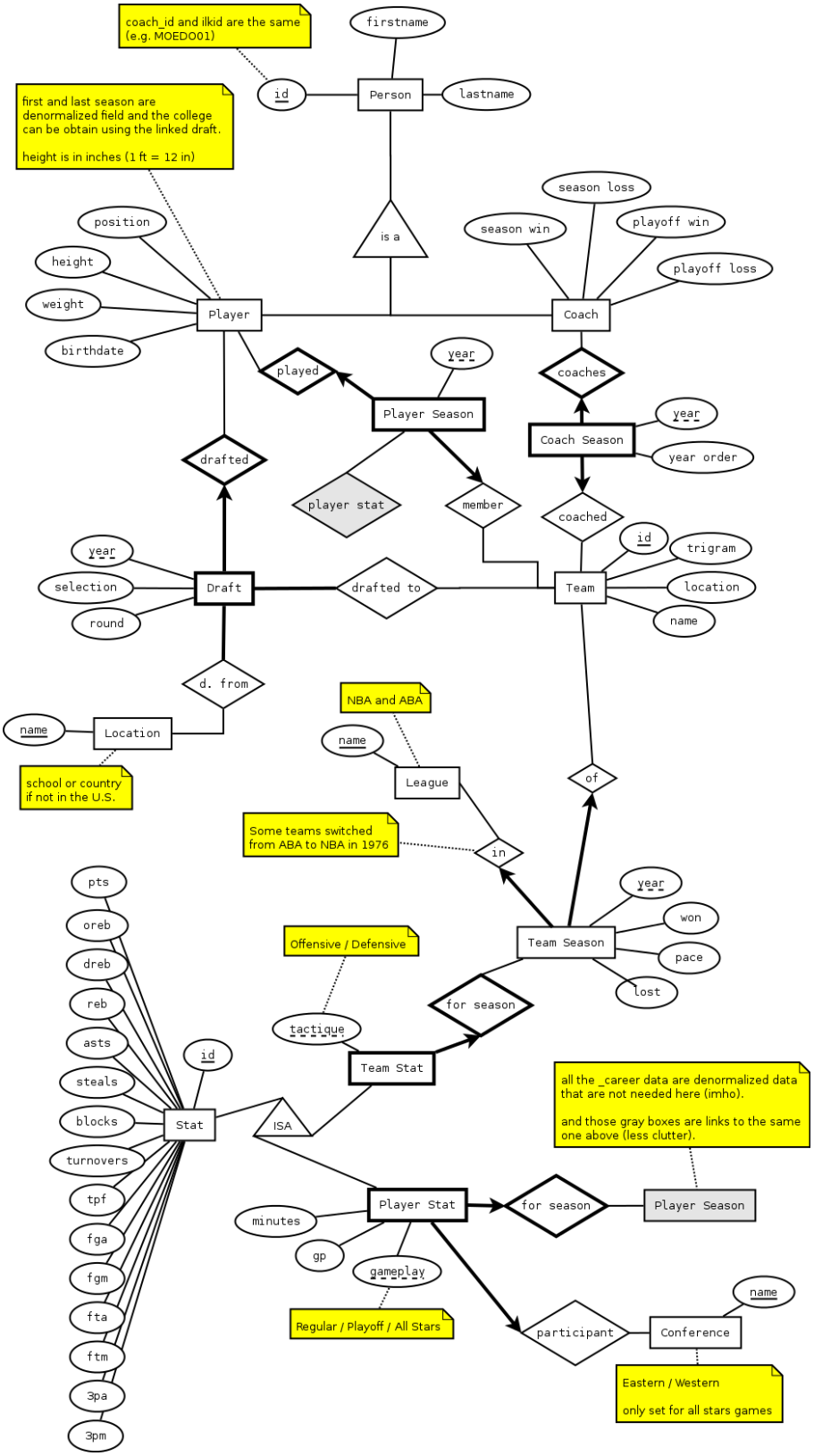


Figure 1.1: ER Schema made using Dia.

Drafts and Location

One major change from the dataset here is that the *People* who go drafted but never played will exist has a *Player*. Those kind of *Player* never played.

Location isn't really important, it's been moved out for further queries.

NB it doesn't have any link with the *Team*'s location which is a city when here, it's the College where the *Player* got drafted from.

Q: *Shouldn't we link Drafts to People and not Players in that case?*

A: No because even players who have never played for an ABA or NBA team have played before for a European team or a school team. Therefore they already have the characteristics of player (position, height, weight, birthdate) when they are drafted.

Stats and Seasons

The *Stats* (statistics) being very standard, it'll live as itself and being kind of *casted* into a *Player* or *Team* stat depending on the case. A *Stat* is all the time linked to a *Season* which is identified by the starting year in the dataset and our model (e.g. *1984* means *the season 84-85*).

Each *Player* and *Team* has a specific *Season* entity linked to it for each *year* played.

Then *Teams* have *offensive* and *defensive* statistics while the *Players* have statistics per kind of *Seasons* played:

- Regular
- Playoff
- and All Stars

NB: All the career stats were seen as denormalized and thus removed. We can get those data back from the yearly *Stats*.

1.2 Design the database and the constraints needed to maintain the database consistent

See the figure on next page.

1.3 Create the SQL commands to create the tables in Oracle

The following SQL schema is really a first shot, with very few constraints on numbers and strings (*varchar*).

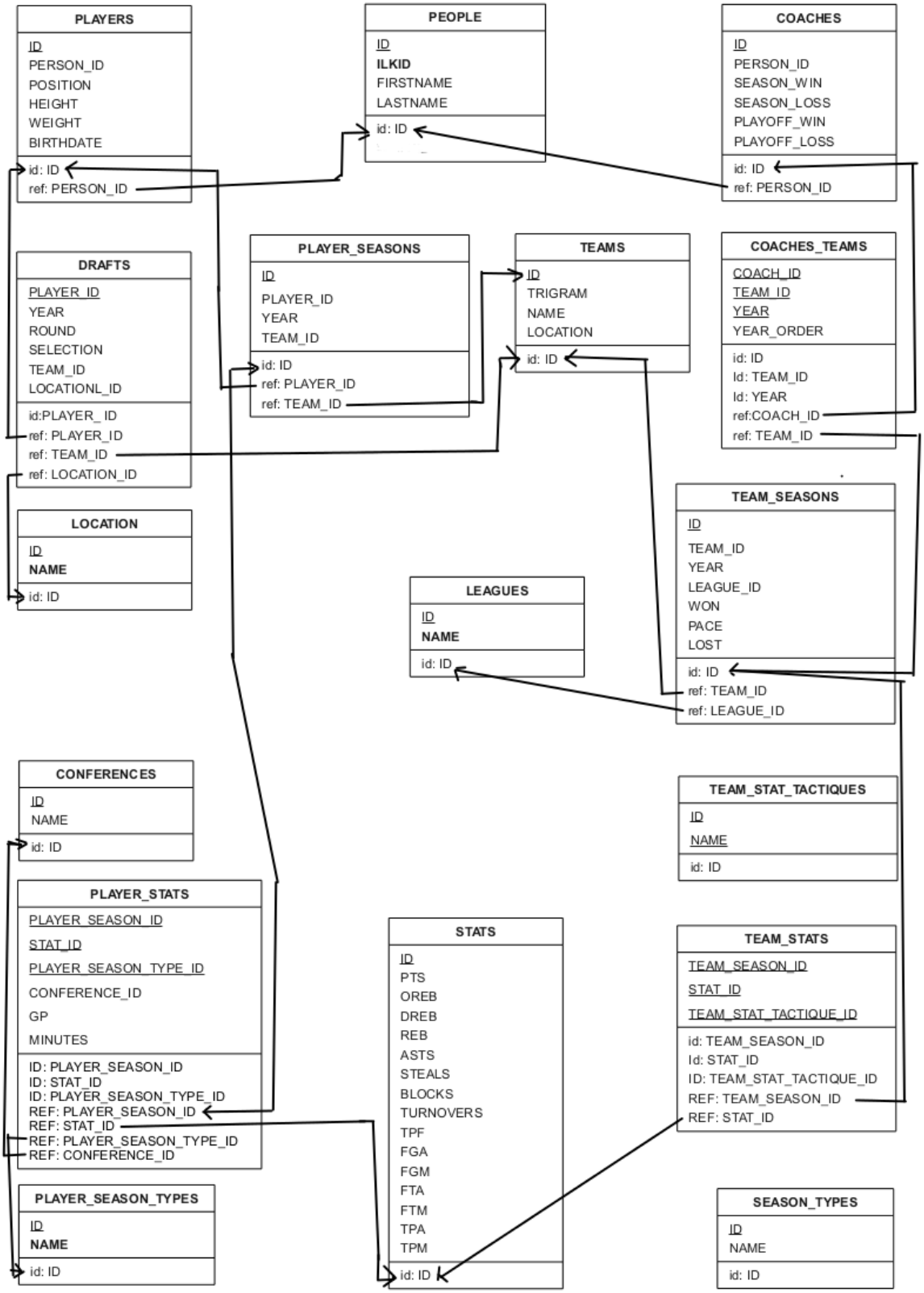
Not being familiar with the way Oracle works, we'll just explain some basic stuff.

Oracle being a very complex RDBMS (*Relational Database Management System*) that we are still learning, this section will simply clarify what we've discovered and wich might explain the following.

It cannot *auto increment* like MySQL or SQLite, so one must use a *sequence* and use it to get the current value or the next value when insert new rows.

```
INSERT INTO leagues (id, name) VALUES (leagues_seq.NEXTVAL, 'NBA');
```

It knows how to delete relations in cascade. It can remove an entire structure from one simple *DELETE*. If *PostgreSQL* can do that has well, the more common *MySQL* or *SQLite* cannot. In the following schema, we activated that cascaded deletion without thinking deeply about it. This schema will be refined in the future when used with the data.


PLAYERS
ID
PERSON_ID
POSITION
HEIGHT
WEIGHT
BIRTHDATE
id: ID
ref: PERSON_ID
PEOPLE
ID
ILKID
FIRSTNAME
LASTNAME
id: ID
COACHES
ID
PERSON_ID
SEASON_WIN
SEASON_LOSS
PLAYOFF_WIN
PLAYOFF_LOSS
id: ID
ref: PERSON_ID
DRAFTS
PLAYER_ID
YEAR
ROUND
SELECTION
TEAM_ID
LOCATIONL_ID
id:PLAYER_ID
ref: PLAYER_ID
ref: TEAM_ID
ref: LOCATION_ID
PLAYER_SEASONS
ID
PLAYER_ID
YEAR
TEAM_ID
id: ID
ref: PLAYER_ID
ref: TEAM_ID
TEAMS
ID
TRIGRAM
NAME
LOCATION
id: ID
COACHES_TEAMS
COACH_ID
TEAM_ID
YEAR
YEAR_ORDER
id: ID
Id: TEAM_ID
Id: YEAR
ref:COACH_ID
ref: TEAM_ID
LOCATION
ID
NAME
id: ID
TEAM_SEASONS
ID
TEAM_ID
YEAR
LEAGUE_ID
WON
PACE
LOST
id: ID
ref: TEAM_ID
ref: LEAGUE_ID
LEAGUES
ID
NAME
id: ID
CONFERENCES
ID
NAME
id: ID
TEAM_STAT_TACTIQUES
ID
NAME
id: ID
PLAYER_STATS
PLAYER_SEASON_ID
STAT_ID
PLAYER_SEASON_TYPE_ID
CONFERENCE_ID
GP
MINUTES
ID: PLAYER_SEASON_ID
ID: STAT_ID
ID: PLAYER_SEASON_TYPE_ID
REF: PLAYER_SEASON_ID
REF: STAT_ID
REF: PLAYER_SEASON_TYPE_ID
REF: CONFERENCE_ID
STATS
ID
PTS
OREB
DREB
REB
ASTS
STEALS
BLOCKS
TURNOVERS
TPF
FGA
FGM
FTA
FTM
TPA
TPM
id: ID
TEAM_STATS
TEAM_SEASON_ID
STAT_ID
TEAM_STAT_TACTIQUE_ID
id: TEAM_SEASON_ID
Id: STAT_ID
ID: TEAM_STAT_TACTIQUE_ID
REF: TEAM_SEASON_ID
REF: STAT_ID
PLAYER_SEASON_TYPES
ID
NAME
id: ID
SEASON_TYPES
ID
NAME
id: ID

```
--
-- People
-- ======
--
-- A person can be a player and/or a coach at different time of
-- her life.
--
-- ilkit can be NULL for drafted only players
--

CREATE TABLE people (
    id NUMBER,
    ilkid VARCHAR(9),
    firstname VARCHAR(255) NOT NULL,
    lastname VARCHAR(255) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (ilkid)
);

CREATE SEQUENCE people_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE players (
    id NUMBER,
    person_id NUMBER NOT NULL,
    position CHAR(1) NOT NULL,
    height NUMBER, -- in inches
    weight NUMBER,
    birthdate DATE,
    PRIMARY KEY (id),
    FOREIGN KEY (person_id)
        REFERENCES people (id) ON DELETE CASCADE
);

CREATE SEQUENCE players_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE coaches (
    id NUMBER,
    person_id NUMBER NOT NULL,
    season_win NUMBER,
    season_loss NUMBER,
    playoff_win NUMBER,
    playoff_loss NUMBER,
    PRIMARY KEY (id),
    FOREIGN KEY (person_id)
        REFERENCES people (id) ON DELETE CASCADE
);

CREATE SEQUENCE coaches_seq
    START WITH 1
    INCREMENT BY 1;

--
-- Group of people
-- ===============
--
-- Teams, leagues and stuff
--

-- NBA/ABA
```

```
CREATE TABLE leagues (
    id NUMBER,
    name CHAR(3) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (name)
);

CREATE SEQUENCE leagues_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE conferences (
    id NUMBER,
    name VARCHAR(31) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (name)
);

CREATE SEQUENCE conferences_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE teams (
    id NUMBER,
    trigram CHAR(3) NOT NULL,
    name VARCHAR(255),
    location VARCHAR(255),
    PRIMARY KEY (id)
);

CREATE SEQUENCE teams_seq
    START WITH
    INCREMENT BY 1;

CREATE TABLE coaches_teams (
    coach_id NUMBER NOT NULL,
    team_id NUMBER NOT NULL,
    year NUMBER,
    year_order NUMBER,
    PRIMARY KEY (coach_id, team_id, year),
    FOREIGN KEY (coach_id)
        REFERENCES coaches (id) ON DELETE CASCADE,
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE
);

--
-- Physical
-- ========
--
-- A school or a country if it's outside the U.S.
--

CREATE TABLE location (
    id NUMBER,
    name VARCHAR(255),
    PRIMARY KEY (id),
    UNIQUE (name)
);

CREATE SEQUENCE location_seq
    START WITH 1
    INCREMENT BY 1;
```

```
--
-- Drafts
--

CREATE TABLE drafts (
    player_id NUMBER NOT NULL,
    year NUMBER NOT NULL,
    round NUMBER NOT NULL,
    selection NUMBER NOT NULL,
    team_id NUMBER NOT NULL,
    location_id NUMBER NULL,
    PRIMARY KEY (player_id),
    FOREIGN KEY (player_id)
        REFERENCES players (id) ON DELETE CASCADE,
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE,
    FOREIGN KEY (l_id)
        REFERENCES location (id) ON DELETE CASCADE
);

--
-- Stats
-- =====
--
-- All the kind of statistical data
--

CREATE TABLE stats (
    id NUMBER,
    pts NUMBER,
    oreb NUMBER,
    dreb NUMBER,
    reb NUMBER,
    asts NUMBER,
    steals NUMBER,
    blocks NUMBER,
    turnovers NUMBER,
    tpf NUMBER,
    fga NUMBER,
    fgm NUMBER,
    fta NUMBER,
    ftm NUMBER,
    tpa NUMBER, -- 3pa
    tpm NUMBER, -- 3pm
    PRIMARY KEY (id)
);

CREATE SEQUENCE stats_seq
    START WITH 1
    INCREMENT BY 1;

--
-- Teams stats
-- ===========

CREATE TABLE team_seasons (
    id NUMBER,
    team_id NUMBER NOT NULL,
    year NUMBER NOT NULL,
    league_id NUMBER NOT NULL,
    won NUMBER,
    pace NUMBER,
```

```
    lost NUMBER,
    PRIMARY KEY (id),
    CONSTRAINT team_season_unique UNIQUE (team_id, year),
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE,
    FOREIGN KEY (league_id)
        REFERENCES leagues (id) ON DELETE CASCADE
);

CREATE SEQUENCE team_seasons_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE team_stat_tactiques (
    id NUMBER NOT NULL,
    name VARCHAR(31),
    PRIMARY KEY (id),
    UNIQUE (name)
);

CREATE SEQUENCE team_stat_tactiques_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE team_stats (
    team_season_id NUMBER NOT NULL,
    stat_id NUMBER NOT NULL,
    team_stat_tactique_id NUMBER NOT NULL,
    PRIMARY KEY(team_season_id, stat_id, team_stat_tactique_id),
    FOREIGN KEY (team_season_id)
        REFERENCES team_seasons (id) ON DELETE CASCADE,
    FOREIGN KEY (stat_id)
        REFERENCES stats (id) ON DELETE CASCADE
);

--
-- Players stats
-- =============

CREATE TABLE player_seasons (
    id NUMBER,
    player_id NUMBER NOT NULL,
    year NUMBER NOT NULL,
    team_id NUMBER NOT NULL,
    PRIMARY KEY (id),
    CONSTRAINT player_season_unique UNIQUE (player_id, year),
    FOREIGN KEY (player_id)
        REFERENCES players (id) ON DELETE CASCADE,
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE
);

CREATE SEQUENCE player_seasons_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE player_season_types (
    id NUMBER,
    name VARCHAR (31),
    PRIMARY KEY (id),
    UNIQUE (name)
);
```

```
CREATE SEQUENCE player_season_types_seq
    START WITH 1
    INCREMENT BY 1;

CREATE TABLE player_stats(
    player_season_id NUMBER NOT NULL,
    stat_id NUMBER NOT NULL,
    player_season_type_id NUMBER NOT NULL,
    conference_id NUMBER NULL, -- for all star games only
    gp NUMBER,
    minutes NUMBER,
    PRIMARY KEY (player_season_id, stat_id, player_season_type_id),
    FOREIGN KEY (player_season_id)
        REFERENCES player_seasons (id) ON DELETE CASCADE,
    FOREIGN KEY (stat_id)
        REFERENCES stats (id) ON DELETE CASCADE,
    FOREIGN KEY (player_season_type_id)
        REFERENCES player_season_types (id) ON DELETE CASCADE,
    FOREIGN KEY (conference_id)
        REFERENCES conferences (id) ON DELETE CASCADE
);
```

1.4 Conclusion

As we decided to go with Ruby on Rails, we expect future changes to be mainly imposed by any limitations that ActiveRecord has.

CHAPTER

TWO

DELIVERABLE 2

The students should accommodate the situation where new data is inserted in any table. Moreover, a simple query which can search for a keyword in any table should be implemented. The user should be able to see more details of the result of the query (e.g., if someone searches for Michael Jordan's regular season statistics and the result has multiple seasons, he/she should be able to see statistics for individual seasons - for example, through a hyperlink).

2.1 Post-mortem deliverable 1

Some of the feedbacks we had.

> *In general, normalizing data is a good thing, but keep in mind that rebuilding some of the information that you discarded might be expensive. Do not hesitate to work with denormalized data if that will make your life easier. That will also help you to have a smaller ER model.*
>
> *On the other hand, working on normalization gave you some great insights on the data. You did a really nice job at studying the domain (i.e., NBA) and exploit such knowledge to fine tune your DB design!*

For the deliverable 2, we didn't had to start denormalizing again but were forced to move data around and denormalizing more in order to gain a better understanding of the dataset. Denormalization adds constraint on data integrity upon changes and we don't what to have to deal with that right now.

But we expect to be forced to start denormalizing fields for the upcoming tasks as things get more complex and intense.

> *[. . .] you did not explain why certain key constraints led to the translation in tables that you will use*

We read your feedbacks on schemas, *ISA* and key constraints but didn't took the time to go back to the design phase. *Getting real* and *hitting walls* mode.

> *Keep up the good work!*

Maybe the only thing we read after all.

2.1.1 Changes in the schema

We made a lot of changes!

NUMBER is not *INT*

The *NUMBER* datatype stores floating numbers. All the *NUMBER* got changed to promper *INT* except for the ones storing real numbers (i.e. *pace*).

ID everywhere

As mentioned in the conclusion of deliverable 1, we expected some changes regarding limitations imposed by the software we are using, ActiveRecord. The first change has been to set up *id* everywhere. The *active record pattern* has trouble working with composed primary keys which is common for tables expressing a *many-to-many* relationship.

```
CREATE TABLE team_stats (
    team_season_id NUMBER NOT NULL,
    stat_id NUMBER NOT NULL,
    team_stat_tactique_id NUMBER NOT NULL,
    PRIMARY KEY(team_season_id, stat_id, team_stat_tactique_id),
    FOREIGN KEY (team_season_id)
        REFERENCES team_seasons (id) ON DELETE CASCADE,
    FOREIGN KEY (stat_id)
        REFERENCES stats (id) ON DELETE CASCADE
);
```

The composite primary key is converted into a unique constraint.

```
CREATE TABLE team_stats (
    id INT,
    team_id INT NOT NULL,
    year INT NOT NULL,
    team_stat_tactique_id INT NOT NULL,
    stat_id INT NOT NULL,
    pace NUMBER NULL,
    PRIMARY KEY (id),
    CONSTRAINT team_stat_unique UNIQUE (team_id, year, team_stat_tactique_id),
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE,
    FOREIGN KEY (team_stat_tactique_id)
        REFERENCES team_stat_tactiques (id) ON DELETE CASCADE,
    FOREIGN KEY (stat_id)
        REFERENCES stats (id) ON DELETE CASCADE
);
```

You can notice that other elements of this table changed. They are explained below.

> *Sidenote:* when using *InnoDB* with MySQL, it's recommended to always have an id because the engine will store the entry on the disk based on that. Specifying it ensure that newest entries are all close to each other. See: http://backchannel.org/blog/friendfeed-schemaless-mysql

Creating *player_allstars*

We initially thought we could aggregate all the *Stat* of one player into a generic table, for the three season types: *Regular*, *Playoff* and *Allstar*. While doing the import, we realized that the later is too different from the others. *Allstar* is **not** linked wia *Team* and thus cannot be tied to a *Player Season* like the two others.

```
CREATE TABLE player_allstars (
    id INT,
    player_id INT NOT NULL,
    stat_id INT NOT NULL,
    conference_id INT NOT NULL,
    year INT NOT NULL,
    gp INT,
    minutes INT,
    PRIMARY KEY(id),
    CONSTRAINT player_allstars_unique UNIQUE (player_id, conference_id, year),
    FOREIGN KEY (player_id)
        REFERENCES players (id) ON DELETE CASCADE,
    FOREIGN KEY (stat_id)
```

```
        REFERENCES stats (id) ON DELETE CASCADE,
    FOREIGN KEY (conference_id)
        REFERENCES conferences (id) ON DELETE CASCADE
);
```

You can notice the same *pattern* describe above with *id* and the unique constraint.

Fixing the *Coaches*

First, we renamed *Coaches Team* to *Coach Season* which makes much more sense.

Secondly, a mistake was made to store. the *season_win*, *season_loss*, ... into the *Coach* itself. This data is the denormalized one and must be into the *Coach Season* instead.

That made us realizing that we had some data duplication. *Team Season* were storing *won*, *pace* and *lost*. *win* and *lost* can be computed using the *Coach Season* and thus got removed from *Team Season*.

```
CREATE TABLE coaches (
    id INT,
    person_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (person_id)
        REFERENCES people (id) ON DELETE CASCADE
);

-- @weak
CREATE TABLE coach_seasons (
    id INT,
    coach_id INT NOT NULL,
    team_id INT NOT NULL,
    year INT NOT NULL,
    year_order INT,
    season_win INT,
    season_loss INT,
    playoff_win INT,
    playoff_loss INT,
    PRIMARY KEY (id),
    CONSTRAINT coach_seasons_unique UNIQUE (coach_id, team_id, year),
    FOREIGN KEY (coach_id)
        REFERENCES coaches (id) ON DELETE CASCADE,
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE
);
```

Moving the *League* where it belongs and dropping *TeamSeason*

We identified *Team* by its *trigram* (those three-letters) after some trouble during the implementation of the import script. The website where our data are coming from (http://http://www.databasebasketball.com/) helped us understanding that it's—in fact—the league identifies the team as well. Knowing that, we moved the *League* from *Team Season* back into *Team* and as the *Team Season* was only containing the information about the *year*, it got moved into the *Team Stat* as well as *pace* which is displayed on the *Offensive* stat on the mother website.

It was also flawed since—to retrieve the league for *Draft*—we had to guess the *Team Season* based on the *Team* and *year* information. And no guarantees are made that that season has (already) been played.

To summarize this change:

- *Team* know which league it belongs to;
- *TeamStats* are referencing a *Team* and contains the information about *year* and *pace*.
- *TeamSeason* is no more.

```
CREATE TABLE teams (
    id INT,
    league_id INT NOT NULL,
    trigram CHAR(3) NOT NULL,
    name VARCHAR(255),
    location VARCHAR(255),
    PRIMARY KEY (id),
    CONSTRAINT team_unique UNIQUE (league_id, trigram),
    FOREIGN KEY (league_id)
        REFERENCES leagues (id) ON DELETE CASCADE
);

-- @weak
CREATE TABLE team_stats (
    id INT,
    team_id INT NOT NULL,
    year INT NOT NULL,
    team_stat_tactique_id INT NOT NULL,
    stat_id INT NOT NULL,
    pace NUMBER NULL,
    PRIMARY KEY (id),
    CONSTRAINT team_stat_unique UNIQUE (team_id, year, team_stat_tactique_id),
    FOREIGN KEY (team_id)
        REFERENCES teams (id) ON DELETE CASCADE,
    FOREIGN KEY (team_stat_tactique_id)
        REFERENCES team_stat_tactiques (id) ON DELETE CASCADE,
    FOREIGN KEY (stat_id)
        REFERENCES stats (id) ON DELETE CASCADE
);
```

2.2 Import the data from the given CSV files into the created database

As mentioned before, we are using *Ruby on Rails* as the application framework in order to build the final application. Below is the very few step required to set it up correctly, as we did it for our local machines. The database configuration is found into the *config/database.yml* file into the *Rails* application (called *nba*). It's set up to be the same credentials for the local and remote (@EPFL) database for simplicity.

2.2.1 Setting up the schema

The following commands will execute the schema creation and loads some initial data (like *leagues*, *conferences*, and so on).

```
$ sqlplus DB2012_G06/DB2012_G06@XE < document/sql/schema.sql
$ sqlplus DB2012_G06/DB2012_G06@XE < document/sql/data.sql
```

2.2.2 Setting up the Rails application

The application dependencies are listed in the *Gemfile* and you should use *bundler* to handle it. It's as easy as the following.

```
$ sudo gem install rails bundler
$ cd nba
$ bundle install
```

2.2.3 Importing the data

The next commands are creating some extra tables (required by the admin interface) and starts parsing the CSV files (from the *dataset* directory).:

```
$ cd nba
$ # locally
$ rake db:migrate
$ rake import:all
$ # remotely
$ RAILS_ENV=production rake db:migrate
$ RAILS_ENV=production rake import:all
```

NB: A faster way of loading data would be to transform the CSV into ready-to-be-inserted CSV and bulk loading them. As we don't trust that much the input data, we went for the much more sluggish approach that performs individual inserts. In the end, it's way easier to debug and it teaches Zen.

2.2.4 Workarounds made to the dataset

We didn't touch the initial file and applied all the workarounds into the import script directly.

Missing teams

Some teams are missing from the CSV, so we are creating them before starting importing everything.

```
missing_teams = {
  "ABA" => %w(NYJ),
  "NBA" => %w(NEW NOR PHW SAN SL1 TAT TOT)
}
missing_teams.each do |league, teams|
  league = League.find_by_name(league)
  teams.each do |trigram|
    puts Team.create(
      :trigram => trigram,
      :league => league
    )
  end
end
```

The Floridians

That team doesn't have any location for historic reasons.

```
# https://en.wikipedia.org/wiki/Miami_Floridians
if row["location"] == "Floridians" then
  row["name"] = row["location"]
  row["location"] = nil
end
```

Non-breaking space

There are some silly characters into the CSV.

```
# One of the coach contains a non-breaking space (nbsp).
row["coachid"] = row[0].tr(" ", " ").strip
```

Magic Johnson and Marques Johnson

Two players are sharing the same *ilkid* and it's a mistake in the data.

```
# Fix a buggy ilkid
if (row["ilkid"] == "JOHNSMA01" and row["firstname"] == "Marques") then
  row["ilkid"] = "JOHNSMA02"
end
```

One entry is a buggy duplicate

In the all star file, there is a duplicate entry that we decided to simply ignore.

```
# Ignore this particular entry
if (row["ilkid"] == "THOMPDA01" and row["year"] == "1982" and row["tpm"] == "NULL") then
  next
end
```

2.3 Accommodate the import of new data in the database they created in the 1st deliverable

To us, this point is about having an admin interface. We've chosen ActiveAdmin which enable easy and nifty automagic interface in respect of the defined model (ActiveRecord).

That's enough for adding simple data. To bulk import more CSV, the import script will just do it by replacing the existing CSV files and running the *import:all* command or any of the per CSV file ones, like: *import:teams*. **NB:** we are assuming it would work but weren't able to test this. You've been warned.

2.3.1 Screenshots

Find below some screenshots of the basic CRUD it enables.

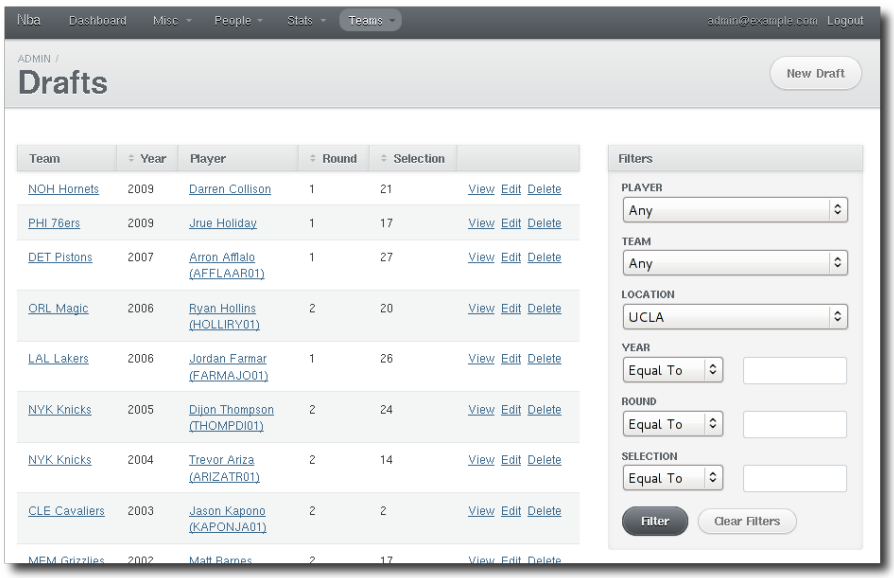


Figure 2.1: Listing all the drafts to NBA made by UCLA.

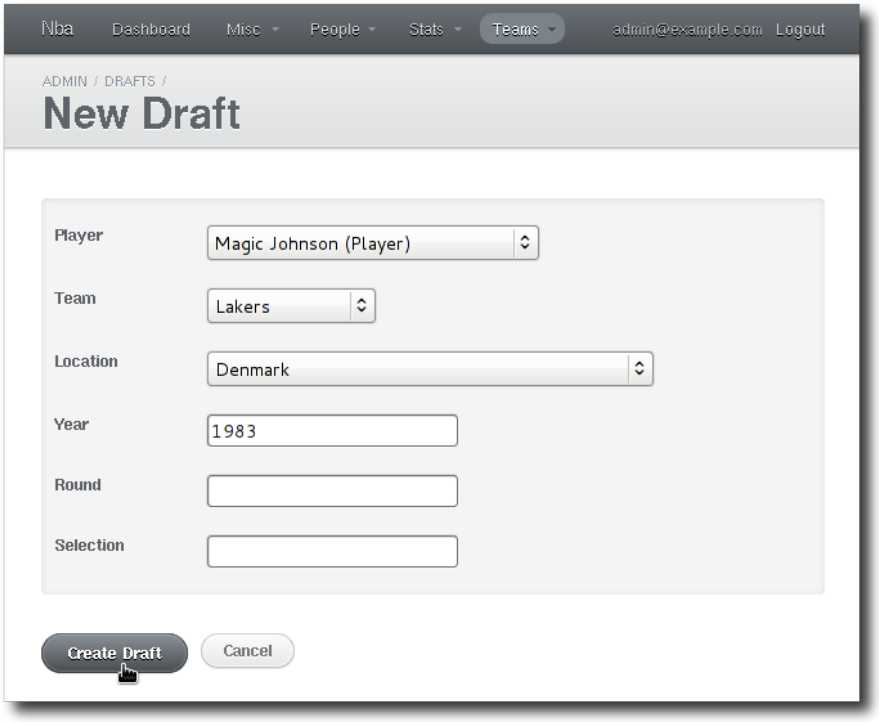


Figure 2.2: Creating a new draft for Magic Johnson.

2.4 Implement the simple search queries

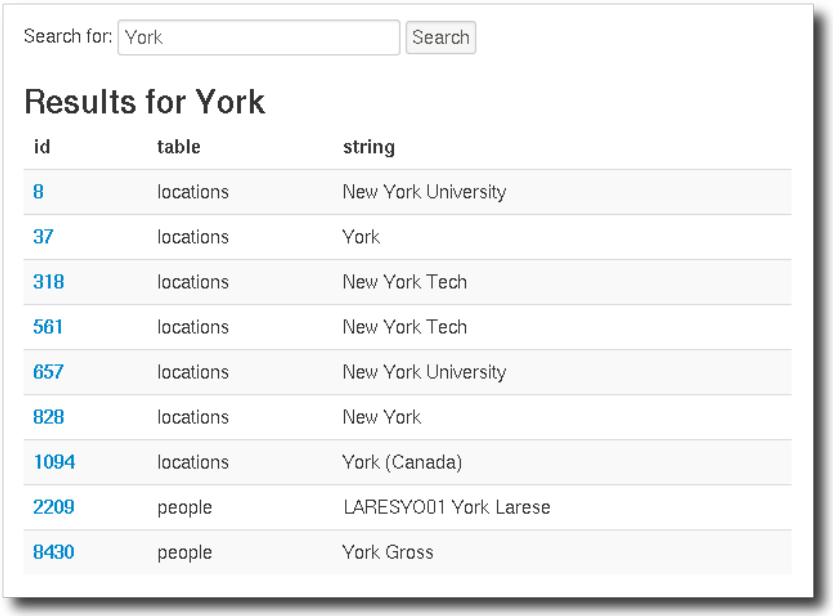
The SQL command is right below. Without any external fulltext search engine, we have to perform a *LIKE '%term%'* on any candidates fields of each tables we would like to be searchable. As it's basically *n* queries, we join them together with the table name to be able to figure out where does it come from.

```
(SELECT
    'teams' tname, id, concat(concat(trigram, ' '), name) str
FROM teams
WHERE
    concat(concat(trigram, ' '), name) LIKE ?

) UNION (

SELECT
    'locations' tname, id,  name str
FROM locations
WHERE name LIKE ?

) UNION (

SELECT
    'people' tname,
    id,
    concat(concat(concat(concat(ilkid, ' '), firstname), ' '), lastname) str
FROM people
WHERE
    concat(concat(concat(concat(ilkid, ' '), firstname), ' '), lastname) LIKE ?

) UNION (

SELECT
    'leagues' tname, id, name str
```

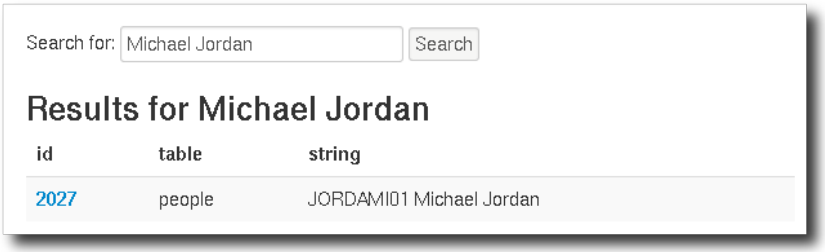
```
FROM leagues
WHERE name LIKE ?

) UNION (

SELECT 'conferences' tname, id, name str
FROM conferences
WHERE name LIKE ?)
```

2.4.1 Screenshots

Find below some screenshot of the search in action. The webpage shows the SQL command ran as well.



Below an example of searching a value made from multiple fields. It's very basic but doing better really requires a better strategy than *LIKE*. There is tons of brilliant softwares that does that very well (Lucene, Solr, Sphinx, Xapian, . . .).



2.4.2 Implement the follow-up search queries of the result of the initial search

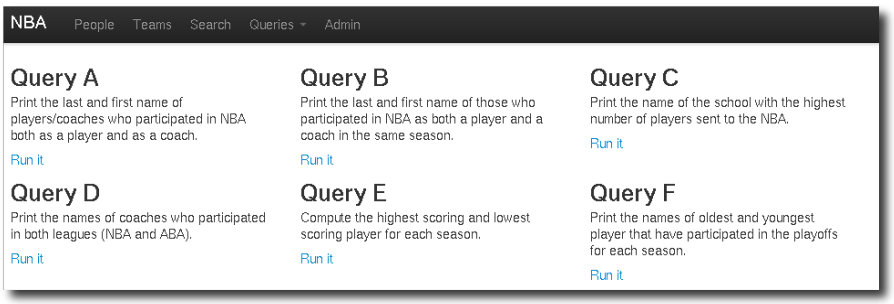
The result of the initial search may look like this.

table name	id	string
people	4050	JAMESMA01 Max Jameson
teams	20	CHI Bulls
. . .	. . .	. . .

From there, we can display something directly and add a link to the proper view for each line.

2.5 Implement using SQL the following queries

A view as been built for each query, which makes them easier to be ran from the web application.



2.5.1 Query A

> *Print the last and first name of players/coaches who participated in NBA both as a player and as a coach.*

The *League* information is tied to a *Team* since each team belongs to a *League*. A *Coach* is linked to a *Team* via the *Coaches Team* table which express a season as coach for the given *Team*. A season played for a *Team* by a *Player* is expressed using the *Player Season* relation. This query fetches all the *Coaches* and all the *Players* from the given *League* and see the ones who match the same *Person*.

```
CREATE OR REPLACE VIEW query_a AS

SELECT DISTINCT
    p.id, p.firstname, p.lastname
FROM
    people p
    JOIN players pl         ON pl.person_id = p.id
    JOIN player_seasons ps  ON ps.player_id = pl.id
    JOIN teams t            ON t.id = ps.team_id
    JOIN leagues l          ON l.id = t.league_id
    JOIN coaches c          ON c.person_id = p.id
    JOIN coach_seasons cs   ON cs.coach_id = c.id
    JOIN teams t2           ON t2.id = cs.team_id
WHERE
    l.name = 'NBA' AND t2.league_id = l.id
ORDER BY
    p.lastname, p.firstname;
```

2.5.2 Query B

> *Print the last and first name of those who participated in NBA as both a player and a coach in the same season.*

This is is very similar to the *Query A* above with one more restriction. The *Coaches Team* and the *Player Season* have to match the same *year* as well.

```
CREATE OR REPLACE VIEW query_b AS

SELECT DISTINCT
    p.id, p.firstname, p.lastname
FROM
    people p
    JOIN players pl         ON pl.person_id = p.id
    JOIN player_seasons ps  ON ps.player_id = pl.id
    JOIN teams t            ON t.id = ps.team_id
```

```
    JOIN leagues l          ON l.id = t.league_id
    JOIN coaches c          ON c.person_id = p.id
    JOIN coach_seasons cs   ON cs.coach_id = c.id
    JOIN teams t2           ON t2.id = cs.team_id
WHERE
    l.name = 'NBA' AND
    l.id = t2.league_id AND
    ps.year = cs.year
ORDER BY
    p.lastname, p.firstname;
```

2.5.3 Query C

Print the name of the school with the highest number of players sent to the NBA.

In order to get the school (or country for foreign players) information, we have to compute among all the drafts which *location* comes first. For people unfamiliar with *Oracle* (like us), you cannot do a simple *GROUP BY locations LIMIT 1* as *LIMIT* doesn't exist in this RDBMS. The alternative is to compute the *RANK()*. The great advantage of RANK is that it may return more than one results in case of equality.

```
CREATE OR REPLACE VIEW query_c AS

SELECT
    id, name, counter
FROM (
    SELECT
        id, name, counter, RANK() OVER (ORDER BY counter DESC) rank
    FROM (
        SELECT
            il.id, il.name, COUNT(il.id) counter
        FROM
            locations il
            JOIN drafts d  ON d.location_id = il.id
            JOIN teams t  ON t.id = d.team_id
            JOIN leagues l ON l.id = t.league_id
        WHERE
            l.name = 'NBA'
        GROUP BY
            il.id, il.name
    )
)
WHERE
    rank = 1;
```

2.5.4 Query D

Print the names of coaches who participated in both leagues (NBA and ABA).

What we are looking for is simply the intersection of coaches who participated in *NBA* with coaches who participated in the *ABA*. In order to facilitate things, we create two views: *nba_coaches* and *aba_coaches*, and use the INTERSECT operator between those views.

The *[an]ba_coaches* views are just simple *JOIN* with a condition on the league name. **NB:** with a different *JOIN* order, it gives no results. With the same order, we have 45 results.

```
CREATE OR REPLACE VIEW nba_coaches AS

SELECT DISTINCT
    p.id, p.lastname, p.firstname
FROM
    teams t
```

```
    JOIN leagues l        ON l.id = t.league_id
    JOIN coach_seasons cs ON cs.team_id = t.id
    JOIN coaches c        ON c.id = cs.coach_id
    JOIN people p         ON p.id = c.person_id
WHERE
    l.name = 'NBA'
ORDER BY
    p.lastname, p.firstname;


CREATE OR REPLACE VIEW aba_coaches AS

SELECT DISTINCT
    p.id, p.lastname, p.firstname
FROM
    teams t
    JOIN leagues l        ON l.id = t.league_id
    JOIN coach_seasons cs ON cs.team_id = t.id
    JOIN coaches c        ON c.id = cs.coach_id
    JOIN people p         ON p.id = c.person_id
WHERE
    l.name = 'ABA'
ORDER BY
    p.lastname, p.firstname;


CREATE OR REPLACE VIEW query_d AS

SELECT * FROM nba_coaches
 INTERSECT
SELECT * FROM aba_coaches;
```

2.5.5 Query E

Compute the highest scoring and lowest scoring player for each season.

By ranking over the number of points a player has gotten, we can easily compute the highest scoring player of all times, by simply ordering by descending rank and querying the one with the first rank.

Since we want the best and worst player for *each* season, we have to use *PARTITION BY year*, which allows us to rank players for each year. Thus, we create two views, one with the best players by year, and one with the worst players. Those views are named *best_players* and *worst_players* respectively.

The next problem is that, for some seasons, there are ex aequos: there might be two or three players that are the best of a season. Similary for worst players, which happens even more often since scoring 0 points is apparently a common occurence. To counter that, we create views based on *best_players* and *worst_players* but with unique years, taking only the last row for every given year.

With all that done, we're just left with combining the *best_players_unique* and *worst_players_unique* views, simply joining them on the year.

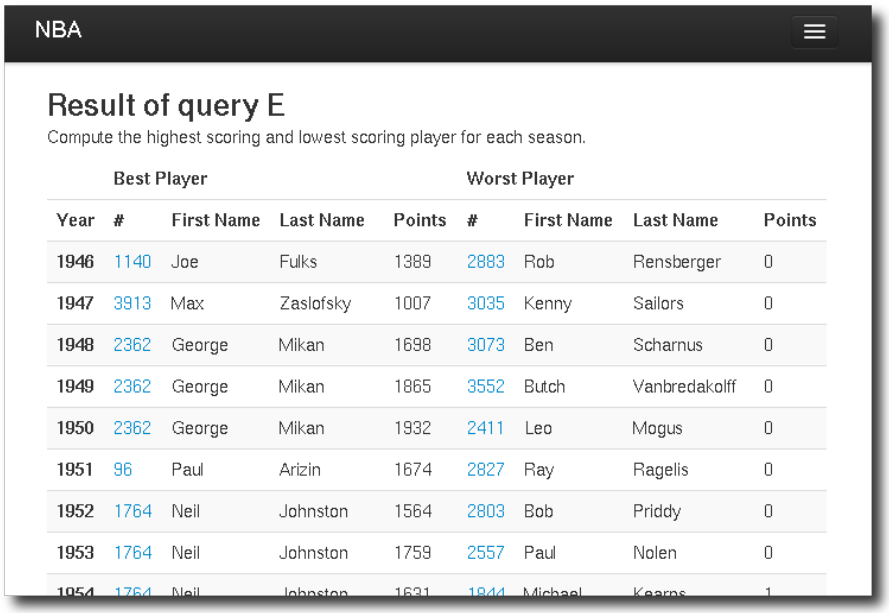
```
CREATE OR REPLACE VIEW best_players AS

SELECT
    player_id best_player_id, player_firstname best_player_firstname,
    player_lastname best_player_lastname, pts best_player_pts, year
FROM (
    SELECT
        player_id, player_firstname, player_lastname, year, pts,
        RANK() OVER (PARTITION BY year ORDER BY pts DESC) r
    FROM (
        SELECT
```

```
            pl.id player_id, p.lastname player_lastname,
            p.firstname player_firstname, psea.year year, stat.pts pts
        FROM
            players pl
            JOIN player_seasons psea ON psea.player_id = pl.id
            JOIN player_stats psta   ON psta.player_season_id = psea.id
            JOIN stats stat          ON psta.stat_id = stat.id
            JOIN people p            ON p.id = pl.person_id
    )
)
WHERE r = 1;


CREATE OR REPLACE VIEW worst_players AS

SELECT
    player_id worst_player_id, player_firstname worst_player_firstname,
    player_lastname worst_player_lastname, pts worst_player_pts, year
FROM (
    SELECT
        player_id, player_firstname, player_lastname, year, pts,
        RANK() OVER (PARTITION BY year ORDER BY pts ASC) r
    FROM (
        SELECT
            pl.id player_id, p.lastname player_lastname,
            p.firstname player_firstname, psea.year year, stat.pts pts
        FROM
            players pl
            JOIN player_seasons psea ON psea.player_id = pl.id
            JOIN player_stats psta   ON psta.player_season_id = psea.id
            JOIN stats stat          ON psta.stat_id = stat.id
            JOIN people p            ON p.id = pl.person_id
    )
)
WHERE r = 1;


CREATE OR REPLACE VIEW best_players_unique AS

SELECT * FROM best_players
WHERE ROWID IN (
    SELECT MAX(ROWID)
    FROM best_players GROUP BY year
);


CREATE OR REPLACE VIEW worst_players_unique AS

SELECT * FROM worst_players
WHERE ROWID IN (
    SELECT MAX(ROWID)
    FROM worst_players
    GROUP BY year
);


CREATE OR REPLACE VIEW query_e AS
SELECT
    bp.year,
    bp.best_player_id, bp.best_player_firstname, bp.best_player_lastname,
    bp.best_player_pts,
    wp.worst_player_id, wp.worst_player_firstname, wp.worst_player_lastname,
    wp.worst_player_pts
```

```
FROM
    best_players_unique bp
    JOIN worst_players_unique wp ON wp.year = bp.year
ORDER BY
    year ASC;
```

Screenshot

Sample output of how it's displayed in the interface.



	Best Player				Worst Player			
Year	#	First Name	Last Name	Points	#	First Name	Last Name	Points
1946	1140	Joe	Fulks	1389	2883	Rob	Rensberger	0
1947	3913	Max	Zaslofsky	1007	3035	Kenny	Sailors	0
1948	2362	George	Mikan	1698	3073	Ben	Scharnus	0
1949	2362	George	Mikan	1865	3552	Butch	Vanbredakolff	0
1950	2362	George	Mikan	1932	2411	Leo	Mogus	0
1951	96	Paul	Arizin	1674	2827	Ray	Ragelis	0
1952	1764	Neil	Johnston	1564	2803	Bob	Priddy	0
1953	1764	Neil	Johnston	1759	2557	Paul	Nolen	0
1954	1764	Neil	Johnston	1631	1844	Michael	Kearns	1

2.5.6 Query F

Print the names of oldest and youngest player that have participated in the playoffs for each season.

This query works exactly like *Query E*, except that instead of *best_players* and *worst_players* we have *youngest_players* and *oldest_players*. The ranking works exactly the same, only ordered by birthdate instead of season points. We also need to weed out duplicates, and to add a test on season types to only get players who participated in the playoffs. Similarly, *youngest_players_unique* and *oldest_players_unique* are joined on year.

```
CREATE OR REPLACE VIEW youngest_players AS

SELECT
    id youngest_id, firstname youngest_firstname, lastname youngest_lastname,
    birthdate youngest_birthdate, year
FROM (
    SELECT
        id, lastname, firstname, year, birthdate,
        RANK() OVER (PARTITION BY year ORDER BY birthdate DESC) r
    FROM (
        SELECT
            pl.id, p.lastname, p.firstname, psea.year, pl.birthdate
        FROM
            players pl
            JOIN player_seasons psea      ON psea.player_id = pl.id
            JOIN player_stats psta        ON psta.player_season_id = psea.id
            JOIN player_season_types psty ON psta.player_season_type_id = psty.id
            JOIN people p ON p.id = pl.person_id
        WHERE
```

```
            psty.name = 'Playoff'
    )
)
WHERE r = 1;


CREATE OR REPLACE VIEW oldest_players AS

SELECT
    id oldest_id, firstname oldest_firstname, lastname oldest_lastname,
    birthdate oldest_birthdate, year
FROM (
    SELECT
        id, lastname, firstname, year, birthdate,
        RANK() OVER (PARTITION BY year ORDER BY birthdate ASC) r
    FROM (
        SELECT
            pl.id, p.lastname, p.firstname, psea.year, pl.birthdate
        FROM
            players pl
            JOIN player_seasons psea      ON psea.player_id = pl.id
            JOIN player_stats psta        ON psta.player_season_id = psea.id
            JOIN player_season_types psty ON psta.player_season_type_id = psty.id
            JOIN people p                 ON p.id = pl.person_id
        WHERE
            psty.name = 'Playoff'
    )
)
WHERE r = 1;


CREATE OR REPLACE VIEW youngest_players_unique AS

SELECT *
FROM youngest_players
WHERE ROWID IN (
    SELECT MAX(ROWID)
    FROM youngest_players
    GROUP BY year
);


CREATE OR REPLACE VIEW oldest_players_unique AS

SELECT *
FROM oldest_players
WHERE ROWID IN (
    SELECT MAX(ROWID)
    FROM oldest_players
    GROUP BY year
);


CREATE OR REPLACE VIEW query_f AS

SELECT
    yp.year,
    yp.youngest_id, yp.youngest_firstname, yp.youngest_lastname,
    yp.youngest_birthdate,
    op.oldest_id, op.oldest_firstname, op.oldest_lastname, op.oldest_birthdate
FROM
    youngest_players yp
    JOIN oldest_players op ON op.year = yp.year
```

```
ORDER BY
    year ASC;
```

2.6 Build an interface to access and visualize the data

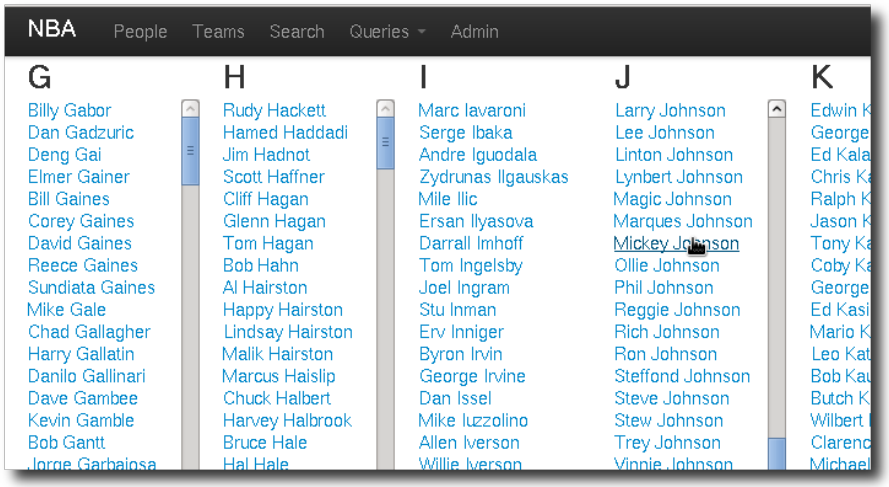
Run the next command to start the webserver.

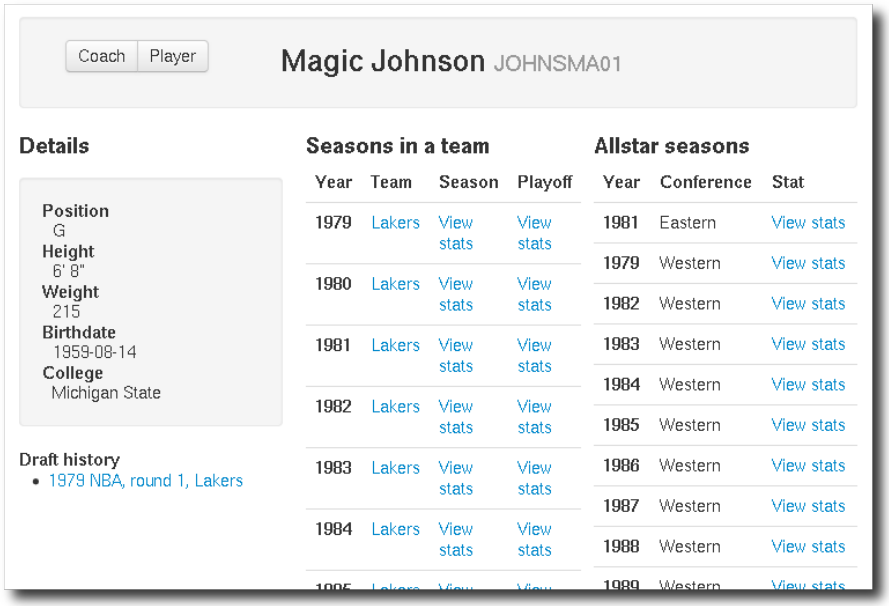
```
$ cd nba
$ # the local database
$ rails server
$ # the remote database at the EPFL
$ rails server -e production
```

2.6.1 Screenshots

Find below some screenshots of the view that exists outside the admin interface. **NB:** they may not reflect the final code.

People





Coach Player

Magic Johnson JOHNSMA01

Details

Position
G
Height
6' 8"
Weight
215
Birthdate
1959-08-14
College
Michigan State

Draft history
- 1979 NBA, round 1, Lakers

Seasons in a team

Year	Team	Season	Playoff
1979	Lakers	View stats	View stats
1980	Lakers	View stats	View stats
1981	Lakers	View stats	View stats
1982	Lakers	View stats	View stats
1983	Lakers	View stats	View stats
1984	Lakers	View stats	View stats

Allstar seasons

Year	Conference	Stat
1981	Eastern	View stats
1979	Western	View stats
1982	Western	View stats
1983	Western	View stats
1984	Western	View stats
1985	Western	View stats
1986	Western	View stats
1987	Western	View stats
1988	Western	View stats
1989	Western	View stats

Coach Player

Magic Johnson JOHNSMA01

Stats

Type	Win	Loss
Overall	5	10
Playoff	0	0

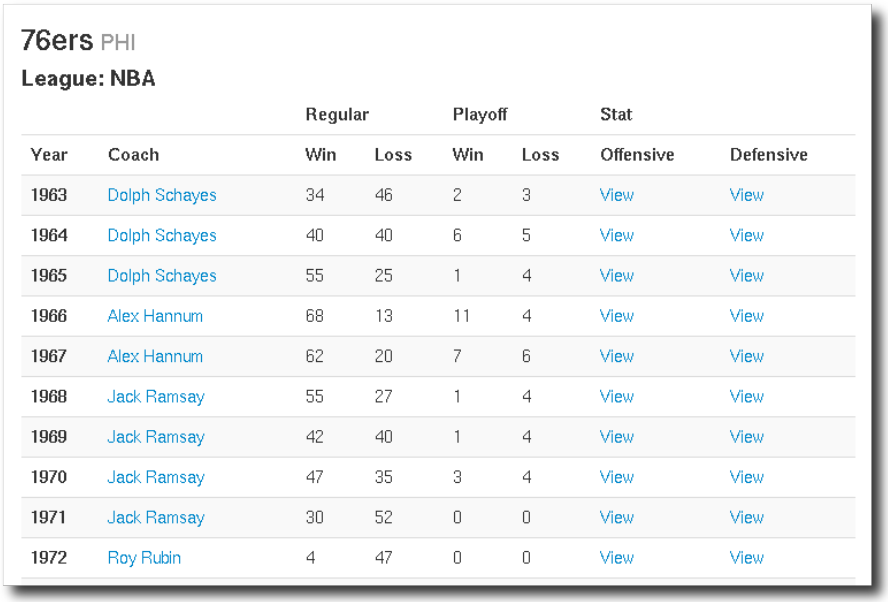
Career

		Regular		Playoff	
Year	Team	Win	Loss	Win	Loss
1993	Lakers	5	10	0	0

Teams

NBA
- Duffey Packers (AND)
- Hawks (ATL)
- Bullets (BA1)
- Bullets (BAL)
- Celtics (BOS)
- Braves (BUF)
- Bullets (CAP)
- Stags (CH1)
- Zephyrs (CH2)
- Packers (CH3)
- Hornets (CHA)
- Bulls (CHI)
- Bobcats (CHR)
- Royals (CIN)
- Rebels (CL1)
- Cavaliers (CLE)
- Mavericks (DAL)
- Falcons (DE1)
- Nuggets (DEN)
- Pistons (DET)
- Nuggets (DN1)
- Zollner Pistons (FTW)

ABA
- Amigos (ANA)
- Cougars (CAR)
- Rockets (DEN)
- Chaparrals (DLC)
- Floridians (FLA)
- Mavericks (HMV)
- Pacers (IND)
- Colonels (KEN)
- Stars (LAS)
- Floridians (MFL)
- Pros (MMP)
- Sounds (MMS)
- Tams (MMT)
- Muskies (MNM)
- Pipers (MNP)
- Americans (NJA)
- Buccaneers (NOB)
- (NYJ)
- Nets (NYN)
- Oaks (OAK)
- Pipers (PTC)
- Pipers (PTP)



76ers PHI

League: NBA

		Regular		Playoff		Stat	
Year	Coach	Win	Loss	Win	Loss	Offensive	Defensive
1963	Dolph Schayes	34	46	2	3	View	View
1964	Dolph Schayes	40	40	6	5	View	View
1965	Dolph Schayes	55	25	1	4	View	View
1966	Alex Hannum	68	13	11	4	View	View
1967	Alex Hannum	62	20	7	6	View	View
1968	Jack Ramsay	55	27	1	4	View	View
1969	Jack Ramsay	42	40	1	4	View	View
1970	Jack Ramsay	47	35	3	4	View	View
1971	Jack Ramsay	30	52	0	0	View	View
1972	Roy Rubin	4	47	0	0	View	View

2.7 Conclusion

We hit the wall pretty hard by getting to know a little better the data while importing it. Guessing out the data really is by reading some CSV is very poor. We guess it's what people call the *implementation gap*. It was way bigger than we usually thought. Knowing that, we can predict that more changes will occur has we progress further.

Working with *Ruby on Rails* and *Oracle XE* is a pretty hard setup to get. But once it works, it's a real bliss (if you have tons of spare RAM). *ActiveAdmin* offers very quickly a way to browse the imported data, spotting mistakes and *ActiveRecord* makes any relationships querying easy.

Oracle is still pretty new to us and the proposed solution are obtained more using the trial and error method than a one shot query that works directly. Stuff like *RANK*, *PARTITION* are uncommon when you have some insight of alternative RDBMS like MySQL, PostgreSQL or SQLite.