Project 4

Justin Hink

Use Case Description

The theoretical use case we'll be examining for this project is as follows:

We are employed by an NHL front office. The management team has hired a consultant who is an expert in simulation techniques to create a team level simulation that aims to estimate playoff probabilities, championship odds and projected standings for the upcoming season (let us assume this is taking place before the current NHL season has started).

Some previous work has been done already by your front office that gives a general idea as to how strong each team in the league is. These projections can be thought of how well each team would do against a schedule of league average teams on neutral ice.

The consultant informs us that he likes the techniques used to come up with these vacuum projections and will be incorporating them into his simulation. He also needs access to the NHL schedule in a convenient format before he can start writing and implementing his projection algorithm.

Part 1: Obtain the Data

The NHL schedule is freely available from the following link:

http://www.nhl.com/ice/scores.htm

There are a number of ways that this can be turned into a convenient to use CSV file. For efficiency, I created a simple .NET app to grab the data, parse it and write it out to a flat file. The relevant code snippet to this application can be found in the code appendix.

The vacuum team projections is sourced from a hobby project of my own. I have them on hand in an .xlsx format so converting them to .csv was a task of minimal effort.

Part 2: Bring the Data into R

Each of the csv files were imported into R using the usual methods. They were saved to R's persistent data format (Rdata) with the save command. See code appendix for this short script. The structure of the two dataframes can be seen in the diagrams below (from RStudio):

Team	GF	GA	G_diff	SF	SA	W	L	OT	P	TeamID
ANA	209.1253	212.5702	-3.444881	2453.099	2514.809	40.42585	31.18061	10.393537	91.24524	1
CGY	195.0109	222.6473	-27.636378	2287.534	2569.174	36.39394	34.20455	11.401516	84.18939	2
EDM	219.0381	224.7955	-5.757371	2569.380	2576.450	40.04044	31.46967	10.489890	90.57077	3
L.A	216.5640	191.6073	24.956642	2540.357	2317.947	45.15944	27.63042	9.210140	99.52902	4
ARI	200.2715	213.0526	-12.781078	2349.242	2517.015	38.86982	32.34763	10.782545	88.52219	5
S.J	216.8702	205.1324	11.737777	2543.949	2395.767	42.95630	29.28278	9.760926	95.67352	6
VAN	204.2353	209.6354	-5.400061	2395.738	2451.126	40.09999	31.42501	10.475003	90.67498	7
CHI	239.6682	204.1859	35.482373	2811.377	2332.791	46.91373	26.31470	8.771568	102.59903	8
COL	211.7079	215.6704	-3.962538	2483.394	2560.872	40.33958	31.24532	10.415106	91.09426	9
DAL	229.0701	212.5393	16.530865	2687.058	2480.759	43.75514	28.68364	9.561214	97.07150	10
MIN	206.4090	217.5584	-11.149466	2421.236	2516.325	39.14176	32.14368	10.714561	88.99807	11
NSH	209.9923	220.7998	-10.807542	2463.269	2512.422	39.19874	32.10094	10.700314	89.09780	12
STL	212.1197	206.0765	6.043235	2488.225	2431.628	42.00721	29.99460	9.998199	94.01261	13
WPG	215.8242	218.7492	-2.925030	2531.679	2484.843	40.51249	31.11563	10.371876	91.39687	14
BOS	223.6854	192.1606	31.524764	2623.893	2405.796	46.25413	26.80940	8.936468	101.44472	15
BUF	190.5697	222.0031	-31.433382	2235.437	2637.260	35.76110	34.67917	11.559724	83.08193	16
DET	202.3947	210.7915	-8.396744	2374.148	2445.884	39.60054	31.79959	10.599864	89.80095	17
FLA	192.8966	215.9062	-23.009529	2262.733	2467.544	37.16508	33.62619	11.208730	85.53889	18
MTL	202.9100	210.3438	-7.433833	2380.192	2527.555	39.76103	31.67923	10.559743	90.08180	19
OTT	211.2474	208.1952	3.052168	2477.992	2460.639	41.50869	30.36848	10.122826	93.14022	20
T.B	226.5781	214.2125	12.365625	2657.826	2481.497	43.06094	29.20430	9.734766	95.85664	21
TOR	208.9852	215.0867	-6.101534	2451.456	2603.583	39.98308	31.51269	10.504231	90.47039	22
CAR	196.8734	218.2015	-21.328107	2309.381	2512.998	37.44532	33.41601	11.138671	86.02930	23
CBJ	216.0003	206.1650	9.835285	2533.745	2495.196	42.63921	29.52059	9.840196	95.11862	24
N.J	202.8045	207.8023	-4.997822	2378.955	2414.884	40.16703	31.37473	10.458243	90.79230	25
NYI	222.2047	210.0325	12.172219	2606.525	2478.020	43.02870	29.22847	9.742824	95.80023	26
NYR	212.9931	203.0893	9.903795	2498.470	2461.440	42.65063	29.51203	9.837342	95.13861	27
PHI	223.1616	222.1133	1.048332	2617.749	2526.664	41.17472	30.61896	10.206319	92.55576	28
PIT	231.7791	218.4675	13.311645	2718.835	2489.722	43.21861	29.08604	9.695348	96.13256	29
WSH	208.2444	209.8698	-1.625412	2442.767	2525.034	40.72910	30.95318	10.317725	91.77592	30

Figure 1- Teamvacuumprojections data frame as seen in RStudio

gameid	Away	AwayID	Home	HomeID
1	MONTREAL	19	TORONTO	22
2	PHILADELPHIA	28	BOSTON	15
3	VANCOUVER	7	CALGARY	2
4	SAN JOSE	6	LOS ANGELES	4
5	COLUMBUS	24	BUFFALO	16
6	NEW JERSEY	25	PHILADELPHIA	28
7	ANAHEIM	1	PITTSBURGH	29
8	MONTREAL	19	WASHINGTON	30
9	BOSTON	15	DETROIT	17
10	FLORIDA	18	TAMPA BAY	21
11	NY RANGERS	27	ST. LOUIS	13
12	OTTAWA	20	NASHVILLE	12
13	CHICAGO	8	DALLAS	10
14	COLORADO	9	MINNESOTA	11
15	CALGARY	2	EDMONTON	3
16	WINNIPEG	14	ARIZONA	5
17	NY ISLANDERS	26	CAROLINA	23
18	WASHINGTON	30	BOSTON	15
19	PITTSBURGH	29	TORONTO	22
20	ANAHEIM	1	DETROIT	17
21	OTTAWA	20	TAMPA BAY	21
22	NEW JERSEY	25	FLORIDA	18
23	CAROLINA	23	NY ISLANDERS	26
24	MONTREAL	19	PHILADELPHIA	28
25	NY RANGERS	27	COLUMBUS	24
26	CALGARY	2	ST. LOUIS	13
27	DALLAS	10	NASHVILLE	12
28	BUFFALO	16	CHICAGO	8
29	MINNESOTA	11	COLORADO	9
30	LOS ANGELES	4	ARIZONA	5
31	EDMONTON	3	VANCOUVER	7
32	WINNIPEG	14	SAN JOSE	6
33	TORONTO	22	NY RANGERS	27
34	WINNIPEG	14	LOS ANGELES	4
35	COLORADO	9	BOSTON	15

Figure 2- First rows of Schedule dataframe as seen in RStudio

Not merging these dataframes before saving to RData was a conscious decision. The data scientist using the data may want to update the team projections on the fly throughout their simulation algorithm (again, we're unsure what method they will be using). Therefore keeping the schedule as its own

distinct entity will allow for much easier updating of that table as needed. If it was not kept separate, think about the case where he might want to downgrade Boston's projection somewhere in his simulation. He'd have to go through the merged data frame, searching for any home OR away instance of Boston and modify the projection. While certainly not possible, it would be annoying code to write if this needed to be done for every team on the fly.

Part 3: Bring the Data into PostgreSQL

For each of the csv files, a table in PostgreSQL was created. See code appendix for this short script. The structure of the simple relational schema is shown in the below diagram.

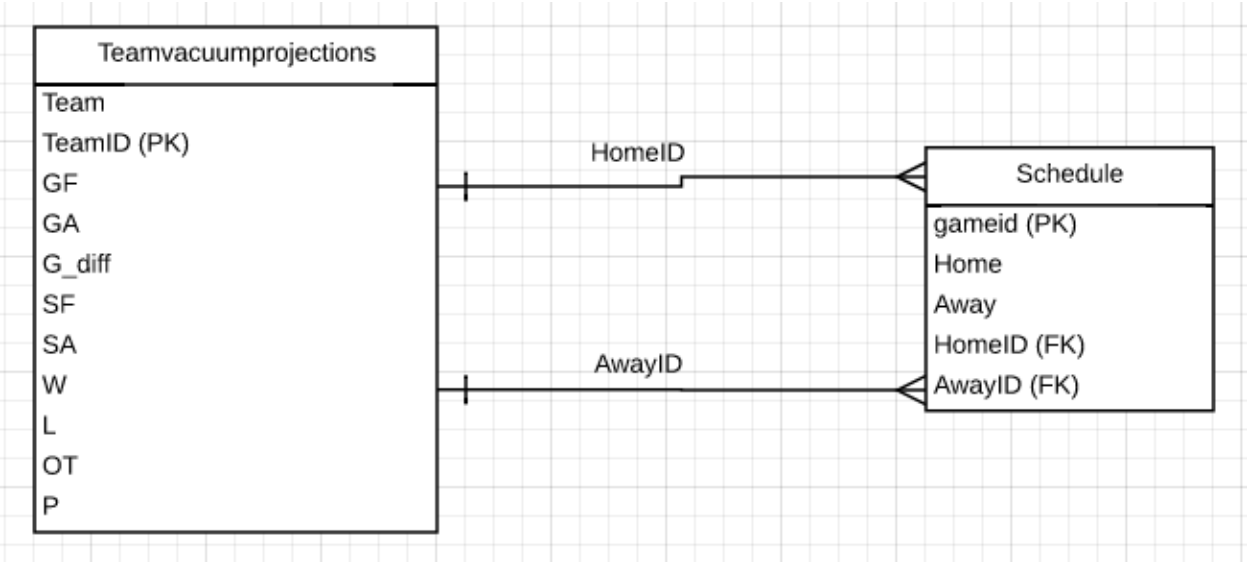

Figure 3-Relational Data Structure

It could be argued quite easily that a 3rd table, "Team" should be created. Team would store the relevant information about a team, not specific to their projection. This would include things like Name, city, state/province official abbreviation etc. Its inclusion would move the "Home" and "Away" columns out of the schedule table. Ditto for "Team" in Teamvacuumprojections. However, due to the limited scope of this exercise, this simple data refactoring task will be left for any future extensions of the project.

With the relational database structure in place, data was loaded into the tables with PGAdmin's handy import from CSV function. The resulting tables can be seen here:

	team character varying(25)	gf double precision	ga double precision	g_diff double precision	sf_diff double precision	sa_diff double precision	w double precision	l double precision	ot double precision	p double precision	teamid integer
1	ANA	209.1252756	212.5701566	-3.444880975	2453.099274	2514.8089	40.42585317	31.18061012	10.39353671	91.24524305	1
2	CGY	195.0109181	222.6472965	-27.63637844	2287.533825	2569.173997	36.39393693	34.2045473	11.40151577	84.18938962	2
3	EDM	219.038126	224.7954971	-5.757371131	2569.379843	2576.450359	40.04043814	31.46967139	10.48989046	90.57076675	3
4	L.A	216.5639643	191.6073223	24.95664203	2540.35722	2317.947267	45.15944034	27.63041975	9.210139915	99.52902059	4
5	ARI	200.2714975	213.0525754	-12.78107792	2349.241926	2517.015473	38.86982035	32.34763474	10.78254491	88.52218561	5
6	S.J	216.8701561	205.1323793	11.73777679	2543.948937	2395.766657	42.95629613	29.2827779	9.760925967	95.67351823	6
7	VAN	204.2352923	209.635353	-5.400060666	2395.738372	2451.126182	40.09998989	31.42500758	10.47500253	90.67498231	7
8	CHI	239.6682367	204.1858633	35.48237336	2811.37694	2332.79118	46.91372889	26.31470333	8.771567777	102.5990256	8
9	COL	211.7078821	215.6704203	-3.9625382	2483.393987	2560.872028	40.33957697	31.24531728	10.41510576	91.09425969	9
10	DAL	229.0701438	212.5392788	16.53086501	2687.058281	2480.758704	43.75514417	28.68364187	9.561213958	97.07150229	10
11	MIN	206.4089575	217.5584232	-11.14946568	2421.236086	2516.324538	39.14175572	32.14368321	10.71456107	88.99807251	11
12	NSH	209.9922563	220.7997984	-10.80754215	2463.269205	2512.421587	39.19874297	32.10094277	10.70031426	89.09780021	12
13	STL	212.1196976	206.0764631	6.043234563	2488.224701	2431.627855	42.00720576	29.99459568	9.99819856	94.01261008	13
14	WPG	215.8241882	218.7492186	-2.925030383	2531.679435	2484.842768	40.51249494	31.1156288	10.37187627	91.39686614	14
15	BOS	223.6853682	192.1606037	31.52476447	2623.893324	2405.795848	46.25412741	26.80940444	8.936468147	101.444723	15
16	BUF	190.5696744	222.0030561	-31.43338167	2235.436767	2637.260099	35.76110305	34.67917271	11.55972424	83.08193034	16
17	DET	202.3947288	210.7914726	-8.396743744	2374.148036	2445.883572	39.60054271	31.79959297	10.59986432	89.80094974	17
18	FLA	192.8966256	215.9061547	-23.00952911	2262.73257	2467.543741	37.16507848	33.62619114	11.20873038	85.53888734	18
19	MTL	202.9100011	210.3438341	-7.433833003	2380.192327	2527.554899	39.76102783	31.67922913	10.55974304	90.08179871	19
20	OTT	211.2473854	208.1952171	3.05216825	2477.992229	2460.63898	41.50869471	30.36847897	10.12282632	93.14021574	20
21	T.B	226.5781309	214.2125056	12.36562526	2657.826257	2481.497213	43.06093754	29.20429684	9.734765614	95.8566407	21
22	TOR	208.9851543	215.0866878	-6.101533523	2451.455611	2603.582745	39.98307775	31.51269169	10.50423056	90.47038606	22
23	CAR	196.8734049	218.2015116	-21.32810665	2309.381329	2512.998177	37.44531556	33.41601333	11.13867111	86.02930223	23
24	CBJ	216.0002823	206.1649976	9.835284698	2533.745068	2495.195661	42.63921412	29.52058941	9.840196471	95.1186247	24
25	N.J	202.80449	207.8023122	-4.99782223	2378.954652	2414.884495	40.16702963	31.37472778	10.45824259	90.79230185	25
26	NYI	222.2047076	210.0324889	12.17221869	2606.524751	2478.019857	43.02870312	29.22847266	9.742824221	95.80023045	26
27	NYR	212.9931203	203.0893253	9.903795027	2498.470199	2461.439736	42.6506325	29.51202562	9.837341874	95.13860688	27
28	PHI	223.1616087	222.1132763	1.048332434	2617.749475	2526.663891	41.17472207	30.61895845	10.20631948	92.55576363	28
29	PIT	231.7791313	218.4674863	13.31164499	2718.835479	2489.722163	43.2186075	29.08604438	9.695348125	96.13256312	29
30	WSH	208.2444224	209.8698345	-1.625412108	2442.766614	2525.03415	40.72909798	30.95317651	10.3177255	91.77592147	30

Figure 4- Teamvacuumprojections PostgreSQL table

	gameid integer	away character varying(25)	awayid integer	home character varying(25)	homeid integer
1	1	MONTREAL	19	TORONTO	22
2	2	PHILADELPHIA	28	BOSTON	15
3	3	VANCOUVER	7	CALGARY	2
4	4	SAN JOSE	6	LOS ANGELES	4
5	5	COLUMBUS	24	BUFFALO	16
6	6	NEW JERSEY	25	PHILADELPHIA	28
7	7	ANAHEIM	1	PITTSBURGH	29
8	8	MONTREAL	19	WASHINGTON	30
9	9	BOSTON	15	DETROIT	17
10	10	FLORIDA	18	TAMPA BAY	21
11	11	NY RANGERS	27	ST. LOUIS	13
12	12	OTTAWA	20	NASHVILLE	12
13	13	CHICAGO	8	DALLAS	10
14	14	COLORADO	9	MINNESOTA	11
15	15	CALGARY	2	EDMONTON	3
16	16	WINNIPEG	14	ARIZONA	5
17	17	NY ISLANDERS	26	CAROLINA	23
18	18	WASHINGTON	30	BOSTON	15
19	19	PITTSBURGH	29	TORONTO	22
20	20	ANAHEIM	1	DETROIT	17
21	21	OTTAWA	20	TAMPA BAY	21

Figure 5- Top 20 Rows of NHL Schedule

Part 4: Bring the Data into MongoDB

Each of the csv files was loaded into MongoDB using the mongoimport utility. This short script can be found in the code appendix at the end of the report.

The data was loaded into 2 logical collections, teamprojs and schedule. The structure of the database and collections can be seen in the following diagrams:

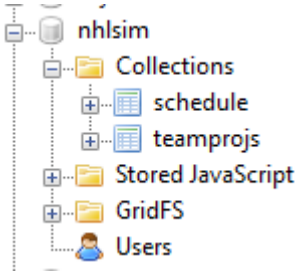

Figure 6- General DataStructure for MongoDB

Json Text of Documents:

```
/* 0 */
{
  "_id" : ObjectId("54551836b48f141766636593"),
  "gameid" : 1,
  "Away" : "MONTREAL",
  "AwayID" : 19,
  "Home" : "TORONTO",
  "HomeID" : 22
}

/* 1 */
{
  "_id" : ObjectId("54551836b48f141766636594"),
  "gameid" : 2,
  "Away" : "PHILADELPHIA",
  "AwayID" : 28,
  "Home" : "BOSTON",
  "HomeID" : 15
}

/* 2 */
{
  "_id" : ObjectId("54551836b48f141766636595"),
  "gameid" : 3,
  "Away" : "VANCOUVER",
  "AwayID" : 7,
  "Home" : "CALGARY",
  "HomeID" : 2
}
```

Figure 7- JSON structure of Schedule Documents

```
{
  "_id" : ObjectId("5455186fb48f141766636a61"),
  "Team" : "ANA",
  "GF" : 209.1252756,
  "GA" : 212.5701566,
  "G_diff" : -3.444880975,
  "SF" : 2453.099274,
  "SA" : 2514.8089,
  "W" : 40.42585317,
  "L" : 31.18061012,
  "OT" : 10.39353671,
  "P" : 91.24524305,
  "TeamID" : 1
}

/* 1 */
{
  "_id" : ObjectId("5455186fb48f141766636a62"),
  "Team" : "CGY",
  "GF" : 195.0109181,
  "GA" : 222.6472965,
  "G_diff" : -27.63637844,
  "SF" : 2287.533825,
  "SA" : 2569.173997,
  "W" : 36.39393693,
  "L" : 34.2045473,
  "OT" : 11.40151577,
  "P" : 84.18938962,
  "TeamID" : 2
}
```


Figure 8- JSON structure of TeamVacuumProj documents

Part 5: Compare Methods

RData

Advantages – If the subsequent simulation algorithm is going to be created in R, importing the two relevant dataframes from an RData file is the easiest method. No additional languages would need to be understood by the consultant (other than R) and he should be able to get going on his work right away.

An advantage specific to our case is that the inherent structure of the data we're dealing with fits very naturally into two data frames.

Disadvantages – If the simulation algorithm is not going to be created in R, extracting data from the RData file is more of a pain. And additional import layer involving 3rd party libraries or

csv/export/import would need to be written, complicating the process. Another large disadvantage of RData is the difficulty of having multiple people collaborate on the same set of persistent data. There are no mechanisms to queue/marshall commands/transactions for RData over a network. Rdata files would have to be accessed on a network file share or emailed back and forth. That's a pretty nightmarish scenario, one that will almost certainly lead to errors in the analysis stage.

PostgreSQL

Advantages – Firstly, the data fits perfectly into a two table relational structure. One join is required but it's as simple as possible. *Seven Databases in Seven Weeks* suggests that PostgreSQL is fantastic for "Stepford data". This data is about as Stepford as possible.

Secondly, PostgreSQL is great for collaboration and if the simulation expert needs us to add a column to one of the data tables, we can in a centralized place without having to send a timestamped flat file back to them.

More a more general advantage of PostgreSQL (and most mature relational databases) is the strength of the tooling available for the platform. We will have little concern that any of the tools our consultant will be using will introduce hard to squash bugs.

Disadvantages - Most of the traditional disadvantages to relational databases aren't very relevant in our scenario. We don't have any need to scale the data architecture out in the near future as this is an internal R&D type project and our data requirements are well defined so the static nature of the schema is something we can deal with. The one clear disadvantage is that going the PostgreSQL route will require the consultant to understand SQL. While that shouldn't be an issue, it could add some time (and $) to the engagement.

One other small disadvantage is that we'll need to give the consultant permissions on our PostgreSQL instance. While securing it should be a trivial issue, it does open us up to a small amount of organizational security risk.

MongoDB

Advantages – All of the collaboration advantages of PostgreSQL apply to MongoDB in this scenario as well. Additionally, MongoDB does add a layer of schema flexibility that could be useful if the consultant gets into his work and realizes that the he needs significantly different (or more) data to best run his simulation.

Disadvantages – MongoDB is a much more specialized data platform and while it is growing in popularity, there's a much better chance that the consultant will not be familiar with how to query data from it's JSON Document structure. We don't have a huge amount of data in our problem and the fantastic scaling abilities of Mongo are not all that relevant to us. Nor is it's ease of use when integrating with a web based application layer.

The disadvantage regarding assigning the consultant DB permissions from the PostgreSQL case also applies to Mongo.

Conclusion and Recommendation

As we've seen throughout the exercise, all 3 data platforms can be used fairly easily to store the data that the consultant will need to run their simulation. However, the pros and cons analysis of each platform does lead to a clear recommended approach.

The familiarity, collaboration ability, tooling of PostgreSQL are all very relevant in this scenario. These advantages, coupled with the inherent structure of our data, make it the easy choice.

Code Appendix

PostgreSQL Table Creation

```
CREATE TABLE TeamVacuumProjections
(
  Team character varying(25) NOT NULL,
  GF float NOT NULL,
  GA float NOT NULL,
  G_diff float NOT NULL,
  SF float NOT NULL,
  SA float NOT NULL,
  W float NOT NULL,
  L float NOT NULL,
  OT float NOT NULL,
  P float NOT NULL,
  TeamID integer NOT NULL,
  CONSTRAINT team_proj_id PRIMARY KEY (TeamID)
)
WITH (
  OIDS=FALSE
);
CREATE TABLE Schedule
(
  gameid int NOT NULL,
  Away character varying(25) NOT NULL,
  Home character varying(25) NOT NULL,
  HomeID int NOT NULL,
  AwayID int NOT NULL,
  FOREIGN KEY (HomeID) REFERENCES TeamVacuumProjections(TeamID),
  FOREIGN KEY (AwayID) REFERENCES TeamVacuumProjections(TeamID),
  CONSTRAINT sched_id PRIMARY KEY (gameid)) WITH(OIDS=FALSE);
```

Import Data into R, Save as RData File

```
setwd("C:/CUNY/IS607/Project4")

schedule <- read.table(file = "schedule.csv", header = TRUE, sep = ",")

teamprojs <- read.table(file = "vacuumTeamProjs.csv", header = TRUE, sep = ",")


save(schedule, teamprojs, file="presimdata.RData")
```

Import Data into MongoDB

```
mongoimport –db nhlsim –collection schedule –type csv –file schedule.csv –headerline

mongoimport –db nhlsim –collection teamprojs –type csv –file vacuumTeamProjs.csv –headerline
```

NHL Season Schedule Scraper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SeasonScheduleImporter
{
    public class DailyScheduleScraper
    {
        public List<ScrapedScheduleGame> GetGames()
        {
            List<ScrapedScheduleGame> games = new List<ScrapedScheduleGame>();

            DateTime endDate = new DateTime(2015, 4, 12);

            DateTime currDate = new DateTime(2014, 10, 8);

            while (currDate < endDate)
            {
                string currDateStr = currDate.ToString("MM'/'dd'/'yyyy");

                var url = "http://www.nhl.com/ice/scores.htm?date=" + currDateStr;

                // HtmlWeb is a helper class to get pages from the web
                var web = new HtmlAgilityPack.HtmlWeb();

                // Create an HtmlDocument from the contents found at given url
                var doc = web.Load(url);

                // Create an XPath to find the `tr` elements which relate to a single
player
                //var xpath = "//table" + "/tbody/tr";//"//table[tfoot/@class='paging']"
+ "/tbody/tr";
                var xpath = "//div/@class='sbGame'";
```

```
                //var gameBoxes = doc.DocumentNode.SelectNodes(xpath);

                var gameBoxes = doc.DocumentNode.Descendants("div").Where(d =>
                        d.Attributes.Contains("class") &&
d.Attributes["class"].Value.Split(' ').Any(b => b.Equals("sbGame")));

                foreach (var box in gameBoxes)
                {
                    try
                    {
                        var away =
box.ChildNodes[2].ChildNodes[1].ChildNodes[1].ChildNodes[0].InnerText;

                        if (away == "Montréal")
                        {
                            away = "Montreal";
                        }

                        var home =
box.ChildNodes[2].ChildNodes[2].ChildNodes[1].ChildNodes[0].InnerText;

                        if (home == "Montréal")
                        {
                            home = "Montreal";
                        }

                        var nhlidSTR =
box.ChildNodes[4].LastChild.ChildNodes[0].Attributes["href"].Value.Split('=')[1];

                        ScrapedScheduleGame scrapedGame = new ScrapedScheduleGame
                        {
                            Date = currDate,
                            Home = home,
                            Away = away
                        };

                        int nhlID = -99;

                        if (Int32.TryParse(nhlidSTR, out nhlID))
                        {
                            scrapedGame.NHLGameID = nhlID;
                        }

                        games.Add(scrapedGame);
                    }
                    catch(Exception ex)
                    {}
                }

                Console.WriteLine("Scraped Games for: " +
currDate.ToString("MM/dd/yyyy"));
                currDate = currDate.AddDays(1);
            }


            return games;
        }
```

```
    }
}


namespace SeasonScheduleImporter
{
    public class ScrapedScheduleGame
    {
        public DateTime Date { get; set; }

        public string Away { get; set; }

        public string Home { get; set; }

        public int NHLGameID { get; set; }
    }
}
```