Django Tango

Internet Baseball Database

Kendall Ahrendsen
Clare Coleman
Deneb Garza
Jordan Graves
Tim Simmons

Introduction

Everyone knows baseball is America's favorite pastime. What most people may not be aware of is the wealth of information and statistics available about the game. Every action on the field is recorded and used to formulate statistics and metrics by which fans and baseball executives may evaluate players and teams. The rich history of baseball is well documented, and many fans enjoy learning about the game's past to better their understanding of its current state.

The Internet Baseball Database (IBD) aims to help those who enjoy reviewing the sport's thoroughly documented history by providing quick and easy access to and navigation of comprehensive data on famous players, current teams, and the recent Major League Baseball seasons. It is a hub for information on historical performance of players, as well as current news, embedded social media, images, and video showcasing the events in a player's career.

Users of the IBD may simply be trying to remember who the Most Valuable Player was three years ago, how a particularly exciting playoff season played out, or perhaps to brush up on their knowledge of their favorite player and his humble beginnings. The IBD is a hub for information on historical performance of players, as well as current news, embedded social media, images, and videos showcasing the events in a player's career. The most common use case of the IBD is someone who is looking for information on a player, team, or season. The IBD displays everything they would ever have wanted to know in an attractive, convenient format.

Models

What are Django Models?

Django models are Python classes that represent a table in a database. An instance of one of these Python classes represents an entry in the table. Each field of the model represents a data field or column of a table. These models serve as a convenient interface into Django's database by allowing the instances of these Python classes to create, retrieve, update or remove the database entries they represent.

Internet Baseball Database Models

There are five models we use to organize our data:

- Player
- Player_Year
- Team
- Team_Year
- Year

Their relationships are as follows:

- Player_Year to Player, many to one
- Player_Year to Year, many to one
- Player_Year to Team_Year, many to one
- Team_Year to Team, many to one
- Team_Year to Year, many to one

Player and Player_Year

The Player model has a composition relationship with the Player_Year model. Each Player model contains information about an individual player that remains persistent throughout a player's career, for example, a player's name, school and handedness. The Player_Year model, on the other hand, holds information about a player which is likely to change from year to year. For example, any yearly statistics, such as batting average or the team to which the player belonged for a specific year, would be attributes of this model.

To establish a link between the Player and Player_Year models, we include in the Player_Year model a ForeignKey to a Player model. This establishes a many to one relationship from Player_Years to Player. Django automatically maintains the backwards relation from Player to Player_Year. By using the option 'related_name' of the ForeignKey, we specify a meaningful name for the backwards relation which allows us to retrieve all Player_Years from a Player.

We want a similar relationship between the Player_Year and Team_Year model so that with an instance of a Team_Year, we can retrieve a set of Player_Year models that represent the players that were on the team represented by that Team_Year model. We establish the same connection with the Year model allowing for us to link to the players' specific statistics for a given year.

With an instance of a Player model we can get all Player_Years that have that Player as a ForeignKey field.

Example:

```
class Player(models.Model):
        ...

class Player_Year(models.Model):
        id = models.IntegerField()
        player = models.ForeignKey(Player, related_name='player_years')

        def(__unicode__):
            return self.id

>>> player_year1 = Player_Year(id=1)
>>> player_year2 = Player_Year(id=2)
>>> player = Player(...)
>>> player.player_years.add(player_year1)
>>> player.player_years.add(player_year2)
>>> player.player_years.all()
>>> [<Player_Year: 1>, <Player_Year: 2>]

or...

>>> player_year3 = Player_Year(id=3, player=player)
>>> player.player_years.all()
>>> [<Player_Year: 1>, <Player_Year: 2>,[<Player_Year: 3>]
```

Team and Team_Year

The Team and Team_Year models are represented in the same way as the Player and Player_Year models, with the Team_Year model containing a ForeignKey field which links to a Team model. Since we will also want a link to individual Team_Years from a Year instance, we do the same for the Year model. The Team_Year to Year relationship allows us to filter all Team_Years by year using the Year model (so that we can display, for example, the team rankings for that year, an attribute held in the Team_Year model. The Team_Year to Team relationship allows us to organize statistics for an individual team by year.

```
class Team_Year(models.Model):

        team = models.ForeignKey(Team, related_name='team_years')

        year = models.ForeignKey(Year, related_name='team_years')
```

Year

The Year model serves as a filter for standings across teams for a particular year and allows us to easily show year-to-year rankings of Teams with links to each Team's Team_Year. This is accomplished by placing the Year model as a Foreign Key inside of a Team_Year which holds that team's ranking for that year.

The Year model also contains Foreign Keys to Player_Year models, which represents the award winners for various awards for that year.

API

The Internet Baseball Database has an JSON API that allows users to programmatically retrieve the information stored in the database. This allows users to develop applications using the IBD easily, avoiding the need to scrape the IBD website for data. It also allows the IBD developers to maintain an idea of how their data is being used by giving out API keys to potential developers, as well as make a small profit when applications that make a certain number of queries are charged accordingly.

/players

The players endpoint of the API contains resources related to Major League Baseball players.

GET	/players
GET	/players/id
POST	/players
PUT	/players/id
DELETE	/players/id

Player Information

id	The IDB assigned identifier for players in the IBD
name	Player names, first and last
social	Twitter handle, if available
number	Jersey number
bats	The handedness of the player when hitting
throws	The handedness of the player when throwing
height	Height, in inches
weight	Weight, in pounds
school	The last school the player attended, could be a high school, college, or international location
years	List of years the player has played professionally in
images	List of images of the player

Example Response

[GET] /players/1

```
Response 200 (application/json)

        {
            "id": "1",
```

```
    "name": "Bryce Harper",
    "social": "@Bharper3407",
    "position": "OF",
    "number": "34",
    "bats": "L",
    "throws": "L",
    "height": "74",
    "weight": "230",
    "school": "College of Southern Nevada",
    "years": ["2012", "2013"],
    "images": ["http://mlb.com/images/players/525x330/547180.jpg"]
}
```


/players/id/years

The player years endpoint of the API contains resources related to Major League Baseball seasons for individual players.

GET	/players/id/years
GET	/players/id/years/id
POST	/players/id/years
PUT	/players/id/years/id
DELETE	/players/id/years/id

Player Year Information

id	The IDB assigned identifier for a player year
year	The year of the season, e.g. 2013
team	Three-letter abbreviation for the team that the player played for, e.g. TEX
type	The type of player; choices include [hitter, pitcher]
games	The number of games in which the player appeared
player_id	The id of the player to whom the year belongs

Hitter Attributes

There are two types of players: hitters and pitchers. They do different things on the field, and thus they have different attributes associated with their performances.

Hitters

pa	Plate appearances: the number of times the hitter took a full turn in the batters box
avg	Batting average: the percentage of plate appearances that the hitter earned a hit
obp	On-base percentage: the batting average, in addition to the percentage o base on balls (walks), that the hitter earned
slg	Slugging percentage: similar to batting average, except that hits for more bases count for extra hits. For example: ● Singles count as one hit ● Doubles count as two hits ● Triples count as three hits ● Home Runs count as four hits
hr	The number of home runs the player hit
rbi	The number of runs batted in that the hitter had

Pitchers

w	Wins: the number of pitching wins a pitcher earned
l	Losses: the number of pitching losses a pitcher suffered
g	Games: the number of games the pitcher appeared in, equal to the number of games started plus games appeared in relief
gs	Games started: the number of games the pitcher started
era	Earned run average: the average number of earned runs scored against a pitcher in nine innings (one game)
s	Saves: credited to a pitcher who finishes a game for the winning team under certain prescribed circumstances.
ip	Innings pitched: the number of innings the player pitched
whip	Walks + Hits Per Inning Pitched

Example Requests/Responses

[GET] players/id/years/2012

Response 200 (application/json)


```
{
    "id": "1",
    "year": "2012",
    "team": "WSN",
    "type": "hitter",
    "games": "139",
    "pa": "597",
    "avg": ".270",
    "obp": ".340",
    "slg": ".477",
    "hr": "22",
    "rbi": "59",
    "player_id": "1"
}
```


/teams

The teams endpoint contains information on the teams that participate in Major League Baseball.

GET	/teams
GET	/teams/id
POST	/teams
PUT	/teams/id
DELETE	/teams/id

Team Information

id	The IDB assigned identifier for a team
name	The team name
abbr	The three-letter abbreviation for the team

city	The city in which the team plays
state	The state in which the team plays
park	The name of the stadium in which the team plays
div	The division of the league in which the team plays, e.g. NL East
social	The Twitter handle of the team
mgr	The team manager
images	A list of images of the team; may include logo, park, etc.
years	The years that the IBD has data for this team

Example Requests/Responses

[GET] teams/id/

Response 200 (application/json)


```
{
            "id": "1",
            "name": "Los Angeles Angels of Anaheim",
            "abbr": "LAA",
            "city": "Los Angeles",
            "state": "CA",
            "park": "Angel Stadium of Anaheim",
            "div": "AL West",
            "social": "@Angels",
            "mgr": "Mike Scioscia",
            "years":
["2004","2005","2006","2007","2008","2009","2010","2011","2012", "2013"],
            "images":
["http://content.sportslogos.net/logos/54/72/thumbs/3zhma0aeq17tktge1huh7yok5.gif"]
          }
```


/teams/id/years

The team years endpoint of the API contains resources related to Major League Baseball seasons for individual teams.

GET	/teams/id/years

GET	/teams/id/years/id
POST	/teams/id/years
PUT	/teams/id/years/id
DELETE	/teams/id/years/id

Team Year Information

id	The IDB assigned identifier for a team year
year	The year of the season, e.g. 2013
wins	The number of games the team won in the regular season
losses	The number of games the team lost in the regular season
standing	The position in which the team finished in the division
playoffs	The playoff finish of the team for a season, empty if they did not make the playoffs
attend	Attendance: the number of people who attended home games during the regular season
payroll	The payroll for players for the year

Example Requests/Responses

[GET] teams/id/years/2013

```
Response 200 (application/json)
{
            "id": "1",
            "year": "2013",
            "wins": "78",
            "losses": "84",
            "standing": "4",
            "playoffs": "Lost LDS (3-2)",
            "attend": "3019505",
            "payroll": "$116032500",
            "team_id": "1"
}
```

/years

The years endpoint of the IBD API contains information about Major League Baseball seasons, and the individual seasons of teams and players who participated in them.

GET	/years
GET	/years/id
PUT	/years/id
DELETE	/years/id

Year Information

id	The IDB assigned identifier for the year
champion	Winner of the World Series for the year
standings	A list of teams sorted by rank for the year
AL_MVP	The American League MVP for the year
NL_MVP	The National League MVP for the year
NL_CY	The National League Cy Young Award winner for the year
AL_CY	The American League Cy Young Award winner for the year

Example Requests/Responses

[GET] year/id/2013

```
Response 200 (application/json)
{
    "id": "2013",
    "champion": "Boston Red Sox",
    "standings": ["BOS", "STL", "OAK", "ATL", "PIT", "DET", "LAD", "CLE", "TBR",
"TEX", "CIN", "WSN", "KCR", "NYY", "BAL", "ARI", "LAA", "SDP", "SFG", "NYM",
"MIL", "TOR", "COL", "PHI", "SEA", "MIN", "CHC", "CHW", "MIA", "HOU"],
    "AL_MVP": "Miguel Cabrera",
    "NL_MVP": "Andrew McCutchen",
```

```
    "AL_CY": "Max Scherzer",
    "NL_CY": "Clayton Kershaw"
}
```

Unit Tests

Unit tests provide a modular means of ensuring that individual components of the IDB API are functioning properly. Each test mimics an HTTP request and verifies that the responses are correct, which will depend upon the request method. There are four cases to consider: GET, POST, PUT, and DELETE. In each case, the test will examine the data returned, which may be treated as JSON objects if there is any, and the response code.

GET

The IDB API makes use of GET in two general circumstances: retrieving a list of entities (players, teams, years), and retrieving a single one of those entities. The two cases are tested similarly; both should have a response code of 200, and then a sampling of the data of a player, team, or year is tested against known values. If the response includes a list of JSON objects, the length of the list can be tested as well as objects with in it.

POST

The IDB API uses POST in the creation of new players, teams, and years. The response code should be 201, indicating a successful creation, and the data returned will be a JSON object that may have a sampling of its data tested against known values.

PUT

The IDB API uses PUT in the modification of a single player, team, or year. The response code

should be 200, indicating a successful query, and the returned data should be a single JSON object whose data may be tested against known values.

DELETE

The IDB API uses DELETE in the removal of a single player, team or year. The response code should be 204, indicating no data. In this instance, attempting to load returned data as JSON will throw an error.