In [1]:
```
!pip install jupysql
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting jupysql
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/1b/48/15b271562ced026001a2ff57833bd78dc92ac97dcc740a9be4f6be849d6a/jupysql-0.10.10-py3-none-any.whl (95 kB)
                                        0.0/95.7 kB ? eta -:--:--
     -------------------------          61.4/95.7 kB 1.7 MB/s eta 0:00:01
     ---------------------------------------- 95.7/95.7 kB 1.4 MB/s eta 0:00:00
Requirement already satisfied: prettytable in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from jupysql) (3.9.0)
Requirement already satisfied: sqlalchemy in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from jupysql) (2.0.27)
Requirement already satisfied: sqlparse in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from jupysql) (0.4.4)
Requirement already satisfied: ipython-genutils>=0.1.0 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from jupysql) (0.2.0)
Requirement already satisfied: jinja2 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from jupysql) (3.1.2)
Collecting sqlglot>=11.3.7 (from jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/33/7b/926f8b57596d45d552b10a30ae11ee70e4c134f60e95fcd04d8ffc2c303e/sqlglot-23.0.5-py3-none-any.whl (378 kB)
                                        0.0/378.7 kB ? eta -:--:--
     -------                           71.7/378.7 kB 4.1 MB/s eta 0:00:01
     --------------------              204.8/378.7 kB 2.5 MB/s eta 0:00:01
     ----------------------            225.3/378.7 kB 2.3 MB/s eta 0:00:01
     ---------------------------------- 378.7/378.7 kB 2.4 MB/s eta 0:00:00
Collecting jupysql-plugin>=0.4.2 (from jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/b3/d9/2f21ca95b8cd62a5cd336c60c2ae271fe3d1928fa38c43c26989013e33d1/jupysql_plugin-0.4.2-py3-none-any.whl (250 kB)
                                        0.0/250.9 kB ? eta -:--:--
     -----------------                  112.6/250.9 kB ? eta -:--:--
     ---------------------------------  225.3/250.9 kB 2.7 MB/s eta 0:00:01
     ---------------------------------- 250.9/250.9 kB 2.6 MB/s eta 0:00:00
Collecting ploomber-core>=0.2.7 (from jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/f4/93/835b343690ceabfde51346c663b7aef2e7211561d916c0a7d6cea32f1415/ploomber_core-0.2.25-py3-none-any.whl (22 kB)
Collecting ploomber-extension (from jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/8e/95/e161f5a0670f44d2ee79f1ca04671186c9cd882542f0572ec5adfc753411/ploomber_extension-0.1.0-py3-none-any.whl (193 kB)
                                        0.0/193.0 kB ? eta -:--:--
     ------------                      61.4/193.0 kB 1.7 MB/s eta 0:00:01
     ---------------------------------- 193.0/193.0 kB 2.3 MB/s eta 0:00:00
Requirement already satisfied: pyyaml in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from ploomber-core>=0.2.7->jupysql) (6.0.1)
Collecting posthog (from ploomber-core>=0.2.7->jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/6c/5f/24cb22118db0e11703b6b80ef9f982eadde21eb585c3a769719e48dce893/posthog-3.5.0-py2.py3-none-any.whl (41 kB)
                                        0.0/41.3 kB ? eta -:--:--
     ---------------------------------- 41.3/41.3 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from jinja2->jupysql) (2.1.3)
Requirement already satisfied: wcwidth in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from prettytable->jupysql) (0.2.6)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from sqlalchemy->jupysql) (4.9.0)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from sqlalchemy->jupysql) (3.0.3)
Requirement already satisfied: requests<3.0,>=2.7 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from posthog->ploomber-core>=0.2.7->jupysql) (2.31.0)
Requirement already satisfied: six>=1.5 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from posthog->ploomber-core>=0.2.7->jupysql) (1.16.0)
Collecting monotonic>=1.5 (from posthog->ploomber-core>=0.2.7->jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/9a/67/7e8406a29b6c45be7af7740456f7f37025f0506ae2e05fb9009a53946860/monotonic-1.6-py2.py3-none-any.whl (8.2 kB)
Collecting backoff>=1.10.0 (from posthog->ploomber-core>=0.2.7->jupysql)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/df/73/b6e24bd22e6720ca8ee9a85a0c4a2971af8497d8f3193fa05390cbd46e09/backoff-2.2.1-py3-none-any.whl (15 kB)
Requirement already satisfied: python-dateutil>2.1 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from posthog->ploomber-core>=0.2.7->jupysql) (2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from requests<3.0,>=2.7->posthog->ploomber-core>=0.2.7->jupysql) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from requests<3.0,>=2.7->posthog->ploomber-core>=0.2.7->jupysql) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from requests<3.0,>=2.7->posthog->ploomber-core>=0.2.7->jupysql) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\k\appdata\local\programs\python\python311\lib\site-packages (from requests<3.0,>=2.7->posthog->ploomber-core>=0.2.7->jupysql) (2023.7.22)
Installing collected packages: monotonic, sqlglot, backoff, posthog, ploomber-core, ploomber-extension, jupysql-plugin, jupysql
Successfully installed backoff-2.2.1 jupysql-0.10.10 jupysql-plugin-0.4.2 monotonic-1.6 ploomber-core-0.2.25 ploomber-extension-0.1.0 posthog-3.5.0 sqlglot-23.0.5
[notice] A new release of pip is available: 23.1.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

CS150A Homework1 - Coding

Instructions / Notes:

Read these carefully

- You will need to install the `jupysql` module to run the scripts. (eg. `pip install --user jupysql`)
- Run the cell below to load the database `hw1.db`.
- You **may** create new Jupyter notebook cells to use for e.g. testing, debugging, exploring, etc.- this is encouraged in fact!
- When you see `In [*]:` to the left of the cell you are executing, this means that the code / query is *running*.
  - **If the cell is hanging- i.e. running for too long: to restart the SQL connection, you must restart the entire python kernel**
  - To restart the kernel using the menu bar: "Kernel >> Restart & Clear Output"), then re-execute the sql connection cell at the top
  - You will also need to restart the connection if you want to load a different version of the database file
- Remember:
  - `%sql [SQL]` is for *single line* SQL queries
  - `%%sql [SQL]` is for *multi line* SQL queries

Submission Instructions:

- Do *NOT* submit your iPython notebook directly.
- Instead, upload your answers in PDF version of the HW1.ipynb with your outputs to Gradescope.

If you hava any confusion, please ask TA team in Piazza.

Have fun!

In [3]:
```
%load_ext sql
%sql sqlite:///hw1.db
```

```
The sql extension is already loaded. To reload it, use:
 %reload_ext sql
```

Understanding the scheme:

In this project we will be working with the commonly-used Lahman baseball statistics database. The database contains pitching, hitting, and fielding statistics for Major League Baseball from 1871 through 2022. It includes data from the two current leagues (American and National), four other "major" leagues (American Association, Union Association, Players League, and Federal League), and the National Association of 1871-1875.

The database is comprised of the following main tables:

- People - Player names, date of birth (DOB), and biographical info
- Batting - batting statistics
- Pitching - pitching statistics
- Fielding - fielding statistics

It is supplemented by these tables:

- AllStarFull - All-Star appearances
- HallofFame - Hall of Fame voting data
- Managers - managerial statistics
- Teams - yearly stats and standings
- BattingPost - post-season batting statistics
- PitchingPost - post-season pitching statistics
- TeamFranchises - franchise information
- FieldingOF - outfield position data
- FieldingPost- post-season fielding data
- FieldingOFsplit - LF/CF/RF splits
- ManagersHalf - split season data for managers
- TeamsHalf - split season data for teams
- Salaries - player salary data
- SeriesPost - post-season series information
- AwardsManagers - awards won by managers
- AwardsPlayers - awards won by players
- AwardsShareManagers - award voting for manager awards
- AwardsSharePlayers - award voting for player awards
- Appearances - details on the positions a player appeared at
- Schools - list of colleges that players attended
- CollegePlaying - list of players and the colleges they attended
- Parks - list of major league ballparks
- HomeGames - Number of homegames played by each team in each ballpark

Read `doc.txt` for more detailed information.

Your Tasks (100 points)

In this part, you should finish task 1 to 4.

We'll use SQL to explore pitching, hitting, and fielding statistics for Major League Baseball from 1871 through 2022.

To start, let's look at the People table in the database we've prepared for you:

In [5]:
```
%%sql
SELECT *
FROM People
LIMIT 1;
```

```
Running query in 'sqlite:///hw1.db'
```

Out[5]:

ID	playerID	birthYear	birthMonth	birthDay	birthCity	birthCountry	birthState	deathYear	deathMonth	deathDay	deathCountry	deathState	deathCit
1	aardsda01	1981	12	27	Denver	USA	CO	None	None	None	None	None	Non

You have now learned how to run SQL statements in a jupyter notebook.

Next please use the following cell to explore any other tables:

In [6]:
```
%%sql
```

```
UsageError: %%sql is a cell magic, but the cell body is empty. Did you mean the line magic %sql (single %)?
```

Alright -- let's get started!

Only need to show the first 10 lines of the results if the lines of results more than 10, i.e., with `LIMIT 10` in the end of your SQL statement.

Task 1: Basics (15 points)

Query 1 (5 points): In the people table, find the `namefirst`, `namelast` and `birthyear` for all players with weight greater than 300 pounds.

In [12]:
```
%%sql
SELECT nameFirst, nameLast, birthYear
FROM People
WHERE weight > 300
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[12]:

nameFirst	nameLast	birthYear
Jumbo	Diaz	1984
Eddie	Gaedel	1925
Walter	Young	1980

Query 2 (5 points): From the people table, group together players born in 1970s (i.e., 1970 ~ 1979) with the same `birthyear`, and report the `birthyear`, average `height`, and number of players for each `birthyear`. Order the results by `birthyear` in ascending order.

In [13]:
```
%%sql
SELECT birthYear, AVG(height) AS averageHeight, COUNT(*) AS numberOfPlayers
FROM People
WHERE birthYear BETWEEN 1970 AND 1979
GROUP BY birthYear
ORDER BY birthYear
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[13]:

birthYear	averageHeight	numberOfPlayers
1970	73.8743169398907	183
1971	73.47619047619048	210
1972	73.38190954773869	199
1973	73.37158469945355	183
1974	73.6774193548387	186
1975	73.47417840375587	213
1976	73.39408866995073	203
1977	73.77560975609757	205
1978	73.6303317535545	211
1979	73.54589371980677	207

Truncated to displaylimit of 10.

Query 3 (5 points): Following the results of Query 2, now only include groups with an average `height` > 73.5. Again order the results by `birthyear` in ascending order.

In [14]:
```
%%sql
SELECT birthYear, AVG(height) AS averageHeight, COUNT(*) AS numberOfPlayers
FROM People
WHERE birthYear BETWEEN 1970 AND 1979
GROUP BY birthYear
HAVING AVG(height) > 73.5
ORDER BY birthYear
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[14]:

birthYear	averageHeight	numberOfPlayers
1970	73.8743169398907	183
1974	73.6774193548387	186
1977	73.77560975609757	205
1978	73.6303317535545	211
1979	73.54589371980677	207

Task 2: Hall of Fame Schools (25 points)

Query 1 (5 points): Find the `namefirst`, `namelast`, `playerid` and `yearid` of all people who were successfully inducted into the Hall of Fame in descending order of `yearid`.

In [18]:
```
%%sql
SELECT P.nameFirst, P.nameLast, H.playerID, H.yearID
FROM HallOfFame H
JOIN People P ON H.playerID = P.playerID
WHERE H.inducted = 'Y'
ORDER BY H.yearID DESC
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[18]:

nameFirst	nameLast	playerID	yearid
Gil	Hodges	hodgegi01	2022
Jim	Kaat	kaatji01	2022
Minnie	Minoso	minosmi01	2022
Tony	Oliva	olivato01	2022
Buck	O'Neil	oneilbu01	2022
David	Ortiz	ortizda01	2022
Derek	Jeter	jeterde01	2020
Ted	Simmons	simmote01	2020
Larry	Walker	walkela01	2020
Harold	Baines	baineha01	2019

Truncated to displaylimit of 10.

Query 2 (10 points): Find the people who were successfully inducted into the Hall of Fame and played in college at a school located in the state of California (i.e., `Schools.state = 'CA'`). For each person, return their `namefirst`, `namelast`, `playerid`, `schoolid`, and `yearid` in descending order of yearid. For this question, yearid refers to the year of induction into the Hall of Fame.

Note: a player may appear in the results multiple times (once per year in a college in California).

In [23]:
```
%%sql
SELECT P.nameFirst, P.nameLast, H.playerID, C.schoolID, H.yearID
FROM HallOfFame H
JOIN People P ON H.playerID = P.playerID
JOIN CollegePlaying C ON H.playerID = C.playerID
JOIN Schools S ON C.schoolID = S.schoolID
WHERE H.inducted = 'Y' AND S.state = 'CA'
ORDER BY H.yearID DESC
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[23]:

nameFirst	nameLast	playerID	schoolID	yearid
Mike	Mussina	mussimi01	stanford	2019
Mike	Mussina	mussimi01	stanford	2019
Trevor	Hoffman	hoffmtr01	cacypre	2018
Trevor	Hoffman	hoffmtr01	cacypre	2018
Randy	Johnson	johnsra05	usc	2015
Randy	Johnson	johnsra05	usc	2015
Randy	Johnson	johnsra05	usc	2015
Pat	Gillick	gillipa99	calavco	2011
Pat	Gillick	gillipa99	usc	2011
Pat	Gillick	gillipa99	usc	2011

Truncated to displaylimit of 10.

Query 3 (10 points): Find the `playerid`, `namefirst`, `namelast` and `schoolid` of all people who were successfully inducted into the Hall of Fame -- whether or not they played in college. Return people in descending order of playerid.

Note: schoolid should be `NULL` or `None` if they did not play in college. Also, this may contains dulplicates results.

In [20]:
```
%%sql
SELECT H.playerID, P.nameFirst, P.nameLast, C.schoolID
FROM HallOfFame H
LEFT JOIN People P ON H.playerID = P.playerID
LEFT JOIN CollegePlaying C ON H.playerID = C.playerID
WHERE H.inducted = 'Y'
ORDER BY H.playerID DESC
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[20]:

playerID	nameFirst	nameLast	schoolID
yountro01	Robin	Yount	None
youngro01	Ross	Youngs	None
youngcy01	Cy	Young	None
yawketo99	Tom	Yawkey	None
yastrca01	Carl	Yastrzemski	None
wynnea01	Early	Wynn	None
wrighha01	Harry	Wright	None
wrighge01	George	Wright	None
winfida01	Dave	Winfield	minnesota
winfida01	Dave	Winfield	minnesota

Truncated to displaylimit of 10.

Task 3: SaberMetrics (30 points)

Query 1 (15 points): Find the `playerid`, `namefirst`, `namelast`, `yearid` and single-year `slg` (Slugging Percentage) of the players with the 10 best annual Slugging Percentage recorded over all time. A player can appear multiple times in the output. For example, if Babe Ruth's `slg` in 2000 and 2001 both landed in the top 10 best annual Slugging Percentage of all time, then we should include Babe Ruth twice in the output. For statistical significance, only include players with more than 50 at-bats in the season. Order the results by `slg` descending, and break ties by `yearid`, `playerid` (ascending).

- Baseball note: Slugging Percentage is not provided in the database; it is computed according to a simple formula you can calculate from the data in the database.
- Note: If you cannot open the Wikipedia link, we have prepared a pdf version of the wiki for you (`Sabermetrics - Wikipedia.pdf` and `Slugging percentage - Wikipedia.pdf`)
- SQL note: You should compute slg properly as a floating point number---you'll need to figure out how to convince SQL to do this!
- Data set note: The online documentation batting mentions two columns 2B and 3B. You can query these columus using backtick, such as `2B` and `3B`.
- Data set note: The column H o f the batting table represents all hits = (# singles) + (# doubles) + (# triples) + (# home runs), not just (# singles) so you'll need to account for some double-counting
- If a player played on multiple teams during the same season (for example anderma02 in 2006) treat their time on each team separately for this calculation

In [33]:
```
%%sql
SELECT P.playerID, Pe.nameFirst, Pe.nameLast, P.yearID,
       (CAST(H - (`2B` + `3B` + HR) + 2 * `2B` + 3 * `3B` + 4 * HR AS FLOAT) / AB) AS slg
FROM Batting P
JOIN People Pe ON P.playerID = Pe.playerID
WHERE AB > 50
ORDER BY slg DESC, P.yearID, P.playerID
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[33]:

playerID	nameFirst	nameLast	yearID	slg
spencsh01	Shane	Spencer	1998	0.9104477611940298
willite01	Ted	Williams	1953	0.9010989010989011
bondsba01	Barry	Bonds	2001	0.8634453781512605
ruthba01	Babe	Ruth	1920	0.849015317286652
ruthba01	Babe	Ruth	1921	0.8462962962962963
bakerje03	Jeff	Baker	2006	0.8245614035087719
anderma02	Marlon	Anderson	2006	0.8125
bondsba01	Barry	Bonds	2004	0.8123324396782842
bondsba01	Barry	Bonds	2002	0.7990074441687345
ruthba01	Babe	Ruth	1927	0.7722222222222223

Truncated to displaylimit of 10.

Query 2 (15 points): Find the `playerid`, `namefirst`, `namelast` and `lslg` (Lifetime Slugging Percentage) for the players with the top 10 Lifetime Slugging Percentage. Lifetime Slugging Percentage (LSLG) uses the same formula as Slugging Percentage (SLG), but it uses the number of singles, doubles, triples, home runs, and at bats each player has over their entire career, rather than just over a single season.

- Note that the database only gives batting information broken down by year; you will need to convert to total information across all time (from the earliest date recorded up to the last date recorded) to compute lslg. Order the results by `lslg` (descending) and break ties by playerid (ascending)
- Note: Make sure that you only include players with more than 50 at-bats across their lifetime.

In [28]:
```
%%sql
SELECT P.playerID, Pe.nameFirst, Pe.nameLast,
       (CAST(SUM(H) - SUM(`2B` + `3B` + HR) + 2 * SUM(`2B`) + 3 * SUM(`3B`) + 4 * SUM(HR) AS FLOAT) / SUM(AB)) AS lslg
FROM Batting P
JOIN People Pe ON P.playerID = Pe.playerID
GROUP BY P.playerID
HAVING SUM(AB) > 50
ORDER BY lslg DESC, P.playerID
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[28]:

playerID	nameFirst	nameLast	lslg
ruthba01	Babe	Ruth	0.6898070969278399
willite01	Ted	Williams	0.6337918505060991
gehrilo01	Lou	Gehrig	0.6324209473815773
foxxji01	Jimmie	Foxx	0.6092943201376936
bondsba01	Barry	Bonds	0.6068853457905962
greenha01	Hank	Greenberg	0.6050452532254958
bassjo01	John	Bass	0.6
tatisfe02	Fernando	Tatis	0.5955598455598455
alvaryo01	Yordan	Alvarez	0.5896084337349398
mcgwima01	Mark	McGwire	0.5881687409083562

Truncated to displaylimit of 10.

Task 4: Salaries (30 points)

Query 1 (10 points): Find the `yearid`, min, max and average of all player salaries for each year recorded in 2010s (i.e., 2010~2019), ordered by `yearid` in ascending order.

In [37]:
```
%%sql
SELECT yearID, MIN(salary) AS minSalary, MAX(salary) AS maxSalary, AVG(salary) AS avgSalary
FROM Salaries
WHERE yearID BETWEEN 2010 AND 2019
GROUP BY yearID
ORDER BY yearID
LIMIT 10;
```

Running query in 'sqlite:///hw1.db'

Out[37]:

yearID	minSalary	maxSalary	avgSalary
2010	400000.0	33000000.0	3278746.825301205
2011	414000.0	32000000.0	3318838.249106079
2012	480000.0	30000000.0	3458421.216981132
2013	480000.0	29000000.0	3723344.353374233
2014	500000.0	26000000.0	3980445.9139650874
2015	507000.0	32571000.0	4301276.094247246
2016	507500.0	33000000.0	4396409.603751466

Query 2 (10 points): Write a query to find the players that had the max salary in 2000 and 2001. Return the `playerid`, `namefirst`, `namelast`, `salary` and `yearid` for those two years. If multiple players tied for the max salary in a year, return all of them.

- Note on notation: you are computing a relational variant of the argmax for each of those two years.

In [43]:
```
%%sql
SELECT P.playerID, Pe.nameFirst, Pe.nameLast, P.salary, P.yearID
FROM Salaries P
JOIN People Pe ON P.playerID = Pe.playerID
WHERE P.yearID IN (2000, 2001) AND P.salary = (
    SELECT MAX(salary)
    FROM Salaries
    WHERE yearID = P.yearID
)
ORDER BY P.yearID, P.salary DESC;
```

Running query in 'sqlite:///hw1.db'

Out[43]:

playerID	nameFirst	nameLast	salary	yearID
brownke01	Kevin	Brown	15714286.0	2000
rodrial01	Alex	Rodriguez	22000000.0	2001

Query 3 (10 points): Each team has at least 1 All Star and may have multiple. For each team in the year 2016, give the `teamid` and diffAvg (the difference between the team's highest paid all-star's `salary` and the team's lowest paid all-star's salary), ordered by diffAvg in descending order.

- Note: Due to some discrepancies in the database, please draw your team names from the All-Star table (so use allstarfull.teamid in the SELECT statement for this).

In [39]:
```
%%sql
SELECT A.teamID, MAX(S.salary) - MIN(S.salary) AS diffAvg
FROM AllstarFull A
JOIN Salaries S ON A.playerID = S.playerID AND A.yearID = S.yearID
WHERE A.yearID = 2016
GROUP BY A.teamID
ORDER BY diffAvg DESC;
```

Running query in 'sqlite:///hw1.db'

Out[39]:

teamID	diffAvg
LAN	32490000.0
NYN	26792671.0
CHN	24473000.0
WAS	17142857.0
BOS	15485500.0
SFN	14577778.0
BAL	14550000.0
TEX	14500000.0
NYA	14492500.0
COL	12428571.0

Truncated to displaylimit of 10.