601.615 Databases Final Project Proposal

Bohao Tang: bhtang@jhu.edu & Shiqing Sun: ssun27@jhu.edu

Nov. 2019

1 Team Members

1. Shiqing Sun
2. Bohao Tang

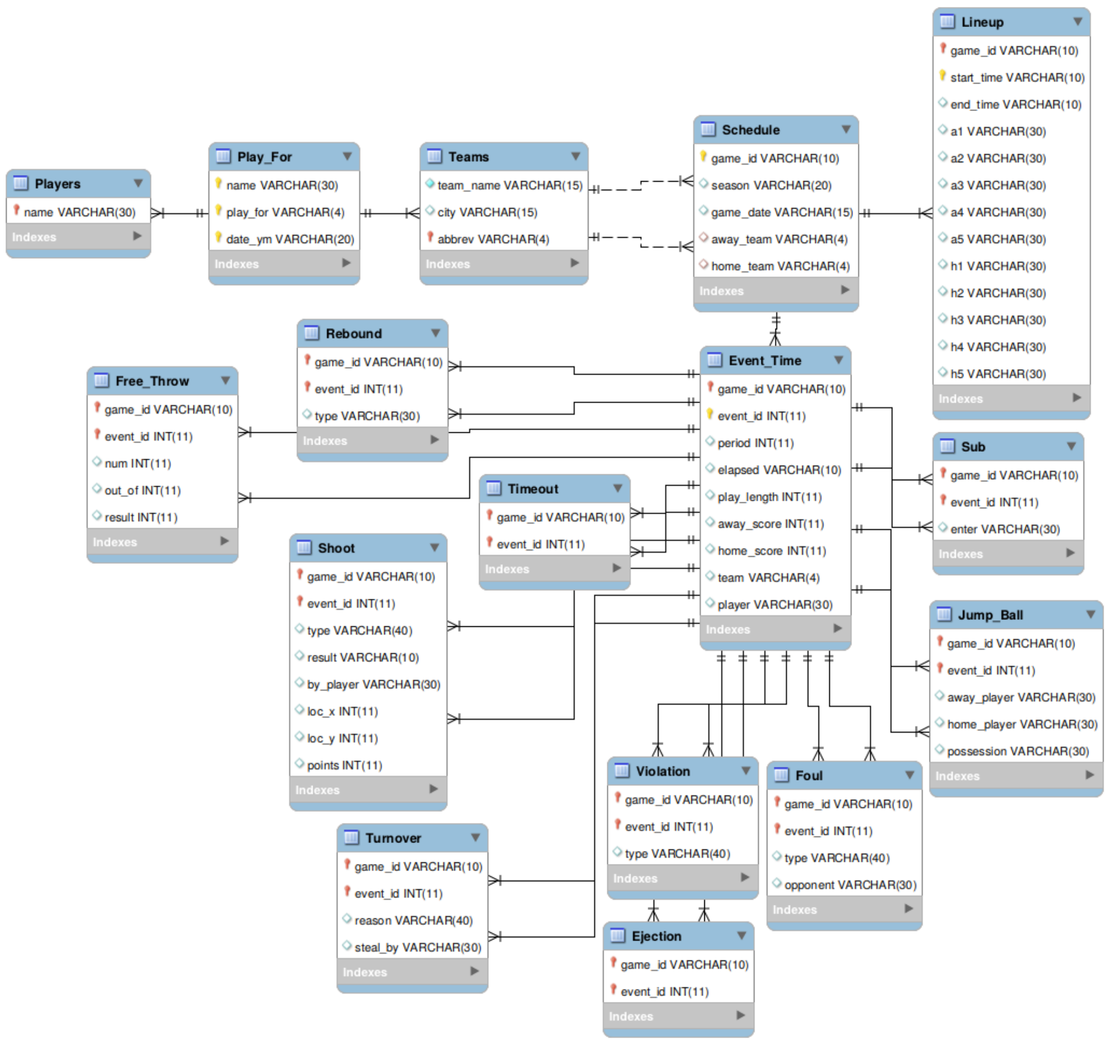
2 Target Domain

We aim to create an NBA play-by-play Database. The raw data contains every event happens in every play in Season 2018-2019.

3 Initial Database Design

Our data are mainly about the log level details of every nba game. By decompositing the raw data, we came out an initial data model shown in picture 3. In the following context, we will explain in detail what do these tables mean and how are they related. Every **table_name** will present a table, and every *attribute* name will present an attribute.

- **Players**: basic information of nba players, currently only *name* information.
- **Teams**: nba team information, like full name *team_name*, *city* they are in and *abbrev* for abbrevation of the team.
- Every **Players** will **Play_For** potential multiple **Teams** and "playing for" information are recorded in every year-month scale as *date_ym*
- Game basic information are recoded in **Schedule** for their *game_id*, *season* played, *game_date*, *away_team* abbrevation and *home_team* abbrevation
- There are multiple **Lineup** (combinations of players playing on court) for every game *game_id* by time, from their *start_time* (48 minute scale) to *end_time*. And the players are *a1*-*a5* for away team and *h1*-*h5* for home team.
- Within every game *game_id* in **Schedule**, their will also be lots of events *event_id*. The time *period*, *elapsed* in every period and *play_length* for every turn are recorded in **Event_Time**, together with *away_score*, *home_score* for scores of away and home teams at that moment. And *team*, *player* for the team and player responsible for that event.



While time information are integrated in **Event_Time** table, different type of event are recorded separatly. They are recorded in following way:

- At *game_id* and *event_id*, *player* in **Event_Time** made a **Shoot** of *type* at location (*loc_x*,*loc_y*). And the *result* will be "made" or "missed" by just himself or "assisted" or "blocked" by *by_player*. Finally the *player* made *points* points.
- At *game_id* and *event_id*, *player* in **Event_Time** got a **Rebound** of *type*.
- At *game_id* and *event_id*, *player* in **Event_Time** got his num_{th} **Free_Throw** out of *out_of* and the *result* will be made or missed.
- At *game_id* and *event_id*, *away_player* and *home_player* had a **Jump_Ball** and finally got by player *possession*.
- At *game_id* and *event_id*, *player* in **Event_Time** made a turnover because of *reason* and if it's stealed, stealed by player *steal_by*.
- At *game_id* and *event_id*, *player* in **Event_Time** made a foul of *type* toward *opponent*.
- At *game_id* and *event_id*, *player* in **Event_Time** made a violation of *type*.
- At *game_id* and *event_id*, *player* in **Event_Time** was ejected in **Ejection**.
- At *game_id* and *event_id*, *team* in **Event_Time** called for a **Timeout**.

- At *game_id* and *event_id*, *player* in **Event_Time** was changed to left the court and player *enter* entered court.

You can find the database creation SQL file in Appendix 1.

4 Sample Questions

The questions can change in phase two, based on other functions the dataset acquire.

1. How many scores did Lebron James make in Season 2018-2019?
2. Who scored the most in Season 2018-2019?
3. What is the Field Goal Percentage of Steph Curry in Season 2018-2019?
4. Who has the highest FG% in Season 2018-2019?
5. Who has the most assist in one game in Season 2018-2019?
6. Who has the most assist in play-off in Season 2018-2019?
7. Who were traded in Season 2018-2019?
8. Which team has the highest win rate in December 2018?
9. Which team is the best first-quarter winner (won the most first quarters)?
10. Which team has the most turnovers in the 4th quarters.
11. What is the most common last name in Season 2018-2019?
12. Which game has the largest score difference within the game in Season 2018-2019?
13. How long is total playing time of James Harden?
14. Which game does James Harden has the highest scoring efficiency (Score/Time)?
15. Who has the highest scoring efficiency in the league?

5 Sample SQL statement

Here we provide the SQL query for the first eight questions.

```
-- a
select sum(points) as result
from shoot
where by_player='LeBron James'
;

-- b
select p.player
from
(select max(total_points) as max_point, player
from
(select sum(points) as total_points, player
from shoot
join event_time
on shoot.game_id=event_time.game_id
and shoot.event_id=event_time.event_id
group by player) as p) as m,
(select sum(points) as total_points , player
from shoot
join event_time
on shoot.game_id=event_time.game_id
and shoot.event_id=event_time.event_id
group by player) as p
where m.max_point = p.total_points
;

-- c
select t.player, (g.good_shots/t.total_shots) as fg
from
(select count(points) as total_shots , player
from shoot
join event_time
on shoot.game_id=event_time.game_id
and shoot.event_id=event_time.event_id
group by player) as t,
(select count(points) as good_shots, player
from shoot
join event_time
on shoot.game_id=event_time.game_id
and shoot.event_id=event_time.event_id
where not points=0
group by player)as g
where t.player=g.player
and t.player='Stephen Curry';


-- 4


select player as result
from
(select max(fg) as max_fg, player
from
(select t.player, (g.good_shots/t.total_shots) as fg
from
(select count(points) as total_shots , player
from shoot
join event_time
on shoot.game_id=event_time.game_id
and shoot.event_id=event_time.event_id
group by player) as t,
(select count(points) as good_shots, player
from shoot
join event_time
on shoot.game_id=event_time.game_id
and shoot.event_id=event_time.event_id
where not points=0
group by player)as g
where t.player=g.player) as fg) as m;

-- 5
select assist, by_player
from
(select max(assist) as max_assist
from
(select count(points) as assist, by_player
from shoot
where not points=0
and not by_player is null
group by by_player, game_id ) as a) as m,
(select count(points) as assist, by_player
from shoot
where not points=0
and not by_player is null
group by by_player, game_id ) as a
where a.assist=m.max_assist;


-- 6
select assist,by_player
from
(select count(points) as assist, by_player
from
(select points, by_player
from schedule,shoot
where game_date}$'2019-04-10'
and schedule.game_id=shoot.game_id
and points }$ 0
and not by_player is null) as s
group by by_player) as a,
(select max(assist) as max_assist
from
(select count(points) as assist, by_player
from
(select points, by_player
from schedule,shoot
where game_date}$'2019-04-10'
and schedule.game_id=shoot.game_id
and points }$ 0
and not by_player is null) as s
group by by_player) as a) as m
where m.max_assist=a.assist;
```

```
-- 7
select distinct a.name
from
(select distinct name,play_for
from play_for) as a,
(select distinct name,play_for
from play_for) as b
where a.name=b.name
and not a.play_for = b.play_for;

-- 8

select win_times/(win_times+lose_times) as win_rate, win_team as team
from
(select count(distinct game_id) as win_times, win_team
from
(select  away_team as lose_team, home_team as win_team, schedule.game_id
from schedule, event_time,
(select max(event_id) as event_id, game_id
from event_time
group by game_id
)as end
where event_time.event_id=end.event_id
and event_time.game_id=end.game_id
and  away_score$\textbf{home_score
and schedule.game_id=end.game_id
union
select * from
(select  away_team as win_team, home_team as lose_team, schedule.game_id
from schedule, event_time,
(select max(event_id)as event_id, game_id
from event_time
group by game_id
)as end
where event_time.event_id=end.event_id
and event_time.game_id=end.game_id
and  away_score}$home_score
and schedule.game_id=end.game_id) as al) as w
group by win_team) as winning,
(select count(distinct game_id) as lose_times, lose_team
from
(select  away_team as lose_team, home_team as win_team, schedule.game_id
from schedule, event_time,
(select max(event_id) as event_id, game_id
from event_time
group by game_id
)as end
where event_time.event_id=end.event_id
and event_time.game_id=end.game_id
and  away_score$\textbf{home_score
and schedule.game_id=end.game_id
union
select * from
(select  away_team as win_team, home_team as lose_team, schedule.game_id
from schedule, event_time,
(select max(event_id)as event_id, game_id
from event_time
group by game_id
)as end
where event_time.event_id=end.event_id
and event_time.game_id=end.game_id
and  away_score}$home_score
and schedule.game_id=end.game_id) as al) as w
group by win_team) as losing
where win_team=lose_team;
```

6 Loading Plan

Our raw data are currently from certificate source (bought online) because stats.nba.com has issues for connection and providing log level data. But we will continue to try if there a way to directly scrap raw data from stats.nba.com.
Given raw data files, we build a formed based interface (a python script) to automatically update data of given "date" into our database. And it can be set to run day by day. You can find the python script in Appendix 2.

7 Output

We plan to make interface to let user search multiple statistics for either NBA players or NBA teams. The statistics can either be general ones including FG% and winning rate, or be special ones related with dynamic process of each game, like which lineup of each team win the most or the distribution of shooting range of each player.
As a part of Data Mining, we want to build a stored procedure to rank and predict the monthly best player given past month data.
And as a part of GUI design, we also want to visualize the so called "heat map" for searched players (that plot out the shooting efficiency of the player at every place on the court). You can see below a possible heatmap 1 from stats.nba.com

8 Advanced Topics

1. Optimization: Optimizing the database design to accelerate certain kind of searching
2. Data Mining: Ranking and Predicting monthly best players
3. GUI: Visualize heat-map of players

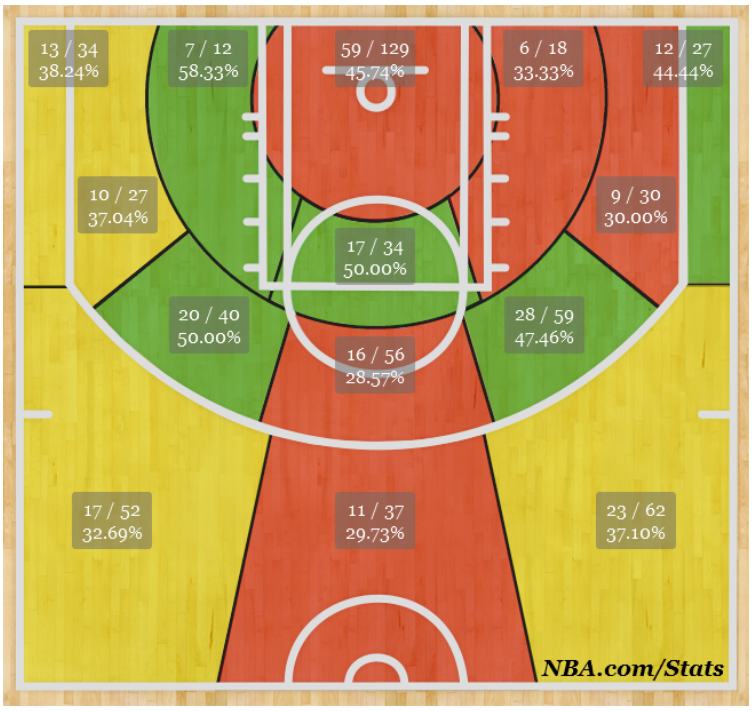


Figure 1: A possible heat map

```
---------------------------------------------------------------
------------------Appendix 1:  Database Creation---------------
---------------------------------------------------------------


use nba;

drop table if exists Players;
create table Players (
    name                VARCHAR(30) not null,
    primary key (name)
);

drop table if exists Play_For;
create table Play_For (
    name                VARCHAR(30),
    play_for            VARCHAR(4),
    date_ym             VARCHAR(20),
    primary key (name, play_for, date_ym)
);

drop table if exists Teams;
create table Teams (
    team_name           VARCHAR(15) not null,
    city                VARCHAR(15),
    abbrev              VARCHAR(4) not null,
    primary key (abbrev)
);

drop table if exists Schedule;
create table Schedule (
    game_id             VARCHAR(10) not null,
    season              VARCHAR(20),
    game_date           VARCHAR(15),
    away_team           VARCHAR(4),
    home_team           VARCHAR(4),
    primary key (game_id)
);

drop table if exists Lineup;
create table Lineup (
    game_id             VARCHAR(10) not null,
    start_time          VARCHAR(10) not null,
    end_time            VARCHAR(10),
    a1                  VARCHAR(30),
    a2                  VARCHAR(30),
    a3                  VARCHAR(30),
    a4                  VARCHAR(30),
    a5                  VARCHAR(30),
    h1                  VARCHAR(30),
    h2                  VARCHAR(30),
    h3                  VARCHAR(30),
    h4                  VARCHAR(30),
    h5                  VARCHAR(30),
    primary key (game_id, start_time)
);

drop table if exists Event_Time;
create table Event_Time (
    game_id             VARCHAR(10) not null,
    event_id            INTEGER not null,
```

```sql
    period              INTEGER,
    elapsed             VARCHAR(10),
    play_length         INTEGER,
    away_score          INTEGER,
    home_score          INTEGER,
    team                VARCHAR(4),
    player              VARCHAR(30),
    primary key (game_id, event_id)
);

drop table if exists Jump_Ball;
create table Jump_Ball (
    game_id             VARCHAR(10) not null,
    event_id            INTEGER not null,
    away_player         VARCHAR(30),
    home_player         VARCHAR(30),
    possession          VARCHAR(30),
    primary key (game_id, event_id)
);

drop table if exists Shoot;
create table Shoot (
    game_id             VARCHAR(10) not null,
    event_id            INTEGER not null,
    type                VARCHAR(40),
    result              VARCHAR(10),
    by_player           VARCHAR(30),
    loc_x               INTEGER,
    loc_y               INTEGER,
    points              INTEGER,
    primary key (game_id, event_id)
);

drop table if exists Rebound;
create table Rebound (
    game_id             VARCHAR(10) not null,
    event_id            INTEGER not null,
    type                VARCHAR(30),
    primary key (game_id, event_id)
);

drop table if exists Turnover;
create table Turnover (
    game_id             VARCHAR(10) not null,
    event_id            INTEGER not null,
    reason              VARCHAR(40),
    steal_by            VARCHAR(30),
    primary key (game_id, event_id)
);

drop table if exists Foul;
create table Foul (
    game_id             VARCHAR(10) not null,
    event_id            INTEGER not null,
    type                VARCHAR(40),
    opponent            VARCHAR(30),
    primary key (game_id, event_id)
);

drop table if exists Violation;
create table Violation (
    game_id             VARCHAR(10) not null,
    event_id          INTEGER not null,
    type              VARCHAR(40),
    primary key (game_id, event_id)
);

drop table if exists Ejection;
create table Ejection (
    game_id           VARCHAR(10) not null,
    event_id          INTEGER not null,
    primary key (game_id, event_id)
);

drop table if exists Timeout;
create table Timeout (
    game_id           VARCHAR(10) not null,
    event_id          INTEGER not null,
    primary key (game_id, event_id)
);

drop table if exists Free_Throw;
create table Free_Throw (
    game_id           VARCHAR(10) not null,
    event_id          INTEGER not null,
    num               INTEGER,
    out_of            INTEGER,
    result            INTEGER,
    primary key (game_id, event_id)
);

drop table if exists Sub;
create table Sub (
    game_id           VARCHAR(10) not null,
    event_id          INTEGER not null,
    enter             VARCHAR(30),
    primary key (game_id, event_id)
);
```

```python
############################################################################
##
##                  Appendix 2: Python Script to Update Database
##
############################################################################


from os import walk
import re
import datetime
import mysql.connector as msc
import pandas as pd

## A class to update nba data
class NBA_Update(object):
    def __init__(self, user, password):
        ## user, password to login to local
        ## database with schema nba
        self.user = user
        self.password = password

    def get_files(self, directory='nba/', date='.*'):
        self.files = []
        for (dirpath, dirnames, filenames) in walk(directory):
            r = re.compile("\[{0}\]".format(date))
            fs = filter(r.match, filenames)
            self.files.extend([dirpath+f for f in fs])

    def compute_time(self, period=1, elapsed = '00:12:00'):
        h, m, s = elapsed.split(':')
        return h + ':' + str(int(m)+12*(period-1)).zfill(2) + ':' + s

    def update_schedule(self, f, data, cursor):
        away, home = re.findall(r'([A-Z]*)@([A-Z]*)',f)[0]
        date = re.findall(r'\[([0-9\-]*)\]',f)[0]
        gameid = re.findall(r'[0-9]+', str(data['game_id'][0]))[0]
        season = re.findall(r'(.*) Season', str(data['data_set'][0]))
        if season:
            season = season[0]
        else:
            season = str(data['data_set'][0])
        add_schedule = ("INSERT IGNORE INTO Schedule "
                        "(game_id, season, game_date, away_team, home_team) "
                        "VALUES (%s, %s, %s, %s, %s) ")
        entry = (gameid, season, date, away, home)
        cursor.execute(add_schedule, entry)
        return

    def update_playfor(self, f, data, cursor):
        away, home = re.findall(r'([A-Z]*)@([A-Z]*)',f)[0]
        date = re.findall(r'\[([0-9\-]*)\-[0-9]+\]',f)[0]
        away_player = list(data['a1'].append(data['a2'])\
                                     .append(data['a3'])\
                                     .append(data['a4'])\
                                     .append(data['a5']).unique())
        home_player = list(data['h1'].append(data['h2'])\
                                     .append(data['h3'])\
                                     .append(data['h4'])\
                                     .append(data['h5']).unique())
        add_playfor = ("INSERT IGNORE INTO Play_For "
                       "(name, play_for, date_ym) "
```

```
                  "VALUES (%s, %s, %s) ")
        for p in away_player:
            cursor.execute(add_playfor, (p,away,date))
        for p in home_player:
            cursor.execute(add_playfor, (p,home,date))
        return

    def update_lineup(self, data, cursor):
        game_id = re.findall(r'[0-9]+', str(data['game_id'][0]))[0]
        add_lineup = ("INSERT IGNORE INTO Lineup "
                     "(game_id, start_time, end_time, "
                     " a1,a2,a3,a4,a5,h1,h2,h3,h4,h5) "
                     "VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s) ")
        lineup = []; timestamp = 0
        for index, row in data.iterrows():
            lu = [row['a1'],row['a2'],row['a3'],row['a4'],row['a5'],\
                  row['h1'],row['h2'],row['h3'],row['h4'],row['h5']]
            if not lu == lineup:
                current = self.compute_time(row['period'], row['elapsed'])
                if not lineup:
                    lineup, timestamp = lu, current
                else:
                    cursor.execute(add_lineup, [game_id, timestamp, current]+lu)
                    lineup, timestamp = lu, current
        current = self.compute_time(row['period'], row['elapsed'])
        cursor.execute(add_lineup, [game_id, timestamp, current]+lu)
        return

    def update_event(self, data, cursor):
        game_id = re.findall(r'[0-9]+', str(data['game_id'][0]))[0]
        add_event_time = ("INSERT IGNORE INTO Event_Time "
                         "(game_id, event_id, period, elapsed, play_length, "
                         " away_score, home_score, team, player) "
                         "VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s) ")
        add_jumpball = ("INSERT IGNORE INTO Jump_Ball "
                       "(game_id, event_id, away_player, home_player, possession)"
                       "VALUES (%s,%s,%s,%s,%s) ")
        add_shoot = ("INSERT IGNORE INTO Shoot "
                    "(game_id, event_id, type, result, "
                    " by_player, loc_x, loc_y, points) "
                    "VALUES (%s,%s,%s,%s,%s,%s,%s,%s) ")
        add_rebound = ("INSERT IGNORE INTO Rebound "
                      "(game_id, event_id, type)"
                      "VALUES (%s,%s,%s) ")
        add_freethrow = ("INSERT IGNORE INTO Free_Throw "
                        "(game_id, event_id, num, out_of, result)"
                        "VALUES (%s,%s,%s,%s,%s) ")
        add_turnover = ("INSERT IGNORE INTO Turnover "
                       "(game_id, event_id, reason, steal_by)"
                       "VALUES (%s,%s,%s,%s) ")
        add_foul = ("INSERT IGNORE INTO Foul "
                   "(game_id, event_id, type, opponent)"
                   "VALUES (%s,%s,%s,%s) ")
        add_violation = ("INSERT IGNORE INTO Violation "
                        "(game_id, event_id, type)"
                        "VALUES (%s,%s,%s) ")
        add_ejection = ("INSERT IGNORE INTO Ejection "
                       "(game_id, event_id)"
                       "VALUES (%s,%s) ")
        add_timeout = ("INSERT IGNORE INTO Timeout "
                      "(game_id, event_id)"
                      "VALUES (%s,%s) ")
        add_sub = ("INSERT IGNORE INTO Sub "
                   "(game_id, event_id, enter)"
                   "VALUES (%s,%s,%s) ")

        for index, row in data.iterrows():
            event_id = int(row['play_id'])
            period, elapsed = int(row['period']), row['elapsed']
            play_length = int(row['play_length'].split(':')[-1])
            away, home = int(row['away_score']),int(row['home_score'])
            team, player = row['team'], row['player']
            if pd.isna(team): team = None
            if pd.isna(player): player = None
            if row['event_type'] == 'timeout':
                team = re.findall(r'(.*) Timeout',row['description'])[0]

            cursor.execute(add_event_time,
                           (game_id,event_id,period,elapsed,\
                            play_length,away,home,team,player))

            if row['event_type'] == 'jump ball':
                a, h, p = row['away'],row['home'],row['possession']
                if pd.isna(a): a = None
                if pd.isna(h): h = None
                if pd.isna(p): p = None
                cursor.execute(add_jumpball,(game_id,event_id,a,h,p))
            elif row['event_type'] == 'shot' or row['event_type'] == 'miss':
                stype, points = row['type'], row['points']
                assist, block, res = row['assist'],row['block'],row['result']
                loc_x,loc_y = row['original_x'],row['original_y']
                if not pd.isna(assist):
                    cursor.execute(add_shoot,
                                   (game_id,event_id,stype,"assisted",\
                                    assist,loc_x,loc_y,points))
                elif not pd.isna(block):
                    cursor.execute(add_shoot,
                                   (game_id,event_id,stype,"blocked",\
                                    block,loc_x,loc_y,points))
                else:
                    cursor.execute(add_shoot,
                                   (game_id,event_id,stype,res,\
                                    None,loc_x,loc_y,points))

            elif row['event_type'] == 'rebound':
                cursor.execute(add_rebound,
                               (game_id,event_id,row['type']))
            elif row['event_type'] == 'free throw':
                cursor.execute(add_freethrow,
                               (game_id,event_id,row['num'],\
                                row['outof'],row['result']))
            elif row['event_type'] == 'turnover':
                steal, r = row['steal'], row['reason']
                if pd.isna(steal): steal = None
                if pd.isna(r): r = None
                cursor.execute(add_turnover, (game_id,event_id,r,steal))
            elif row['event_type'] == 'foul':
                t, o = row['type'],row['opponent']
                if pd.isna(t): t = None
                if pd.isna(o): o = None
                cursor.execute(add_foul,(game_id,event_id,t,o))
            elif row['event_type'] == 'violation':
                cursor.execute(add_violation,
                               (game_id,event_id,row['type']))
            elif row['event_type'] == 'ejection':
                cursor.execute(add_ejection, (game_id, event_id))
            elif row['event_type'] == 'timeout':
                cursor.execute(add_timeout, (game_id, event_id))
            elif row['event_type'] == 'sub':
                cursor.execute(add_sub, (game_id, event_id, row['entered']))
        return

    def update_by_date(self, directory='nba/', date='.*'):
        ## update given date data in directory
        dbx = msc.connect(user=self.user, password=self.password,
                          host='127.0.0.1', database='nba')
        cursor = dbx.cursor()
        self.get_files(directory, date)
        try:
            for (i,f) in enumerate(self.files):
                if i % 100 == 0: print("num_files: ", i)
                data = pd.read_csv(f, encoding = 'latin1')
                self.update_schedule(f, data, cursor)
                self.update_playfor(f, data, cursor)
                self.update_lineup(data, cursor)
                self.update_event(data, cursor)
            dbx.commit()
            print("Success!")
        finally:
            cursor.close()
            dbx.close()

if __name__ == 'main':
    ## update all files
    ex = NBA_Update('bohao', '3316')
    ex.update_by_date()
```