**Problem Description/Project Purpose:**

The ultimate purpose of this project is to provide a visualization to display key metrics (both live and historical) which are indicative of a player’s contribution (whether positive or negative) to their team’s win/loss result in a game. Coaches use these types of analytics to make decisions before, during and after each game in hopes of leading their teams to success. Fans use these types of analytics for many purposes, including ranting on twitter while blaming coaches that make way too much money to make bad decisions.

One coaching decision topic that is hotly debated/tweeted by fan bases is based on player minutes (i.e. The number of minutes each player is given to play during a game). One metric that is popular in fan rants is the “Plus-Minus.”

The plus-minus metric in basketball is a statistic that measures the point differential when a player is on the court. Essentially, it tracks how the team performs in terms of scoring while the player is playing compared to when they're not. Here's a quick breakdown:

* **Positive Plus-Minus**: Indicates that the team scored more points than the opponent while the player was on the court.
* **Negative Plus-Minus**: Indicates that the team was outscored by the opponent while the player was on the court.

For example, if a player has a +5 plus-minus, it means their team scored 5 more points than the opponent while that player was on the floor. Conversely, a -3 means their team was outscored by 3 points during their time on the court. Note that players periodically transition from playing to sitting on the bench to playing again, etc. **The plus-minus metric can only be calculated correctly by accounting for which players are on the court during each change in score during the entire game.**

This metric, while imperfect, is valuable because it provides a more holistic view of a player's impact beyond just their individual stats (like points scored, rebounds, etc.). It's often used by coaches and analysts to assess a player's overall contribution to the team's success.

There are websites and apps dedicated to the calculation of these metrics, but each has their own distinct, undesirable flair. Most of the good ones require a paid subscription. The free ones are buggy and/or littered with unwanted adds. Few offer the customizability to meet the “one-stop-shop” desires of the individual fan while providing a simple, fan-focused analytical experience.

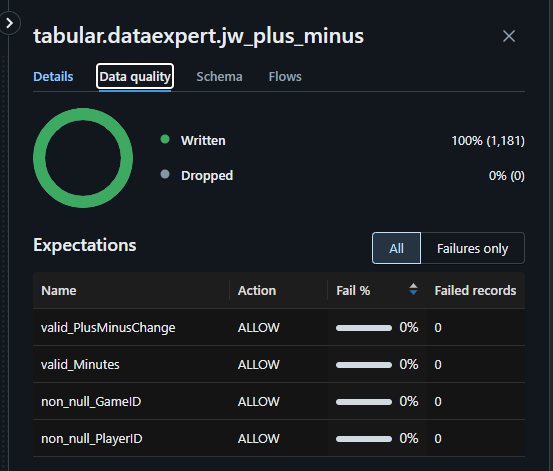
Personal motivations also should be disclosed. As a fan of Illinois basketball, I’ve observed that in each of the past 5 seasons, there is always one player that the coach seems to show favoritism towards and thus award that player more minutes than deserved. This project provides a free, customizable analytical vehicle for me to either:

* Rant more intelligently. This project can add data-enriched fuel to my unquenchable flame of complaints re: player minute decisions
* Relax more. Stop being such a crazy/angry fan. Occasionally. If the data convinces me that my intuition is wrong, I can (theoretically) be at peace with the basketball universe. And I can do it in near-real-time!

**Project Scope:**

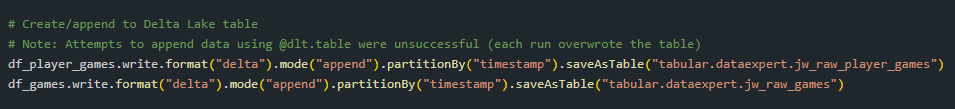
* Given the allotment of 2 weeks to complete this project, I chose to **focus singularly on the plus-minus metric** for the upstream dashboard**.**
* Scope will include **2 primary analysis components: historical data and live data**.
  + Note: Different and completely independent sources will be used for live vs. historical data.
* Data infrastructure will be built to enable future expansion and customization of metrics, but in order to meet 3/2/25 DataExpert.IO bootcamp deadlines, scope of the resulting dashboard will be limited. This is a single-person project and scope creep will not be tolerated!

**Data sources and tech stack:**

* General notes/constraints:
  + I made a choice not to pay for my data sources. Why?
    - I’m generally frugal by nature.
    - I wanted to add a fun challenge. After investigating free trials for a few sites (e.g. [Sportradar.com](https://sportradar.com/)) and unsuccessfully haggling for special “personal use only” access with others, I decided to accept the challenge of somehow doing this for free (as well as legally and ethically).
  + No web scraping. I’m not an expert web scraper and didn’t want to spend time becoming one for this project. For both reasons of efficiency/reliability and ethics, I chose to shun web scraping.
* Data sources
  + Live data: [SportsData.io](https://sportsdata.io/)
    - Notes:
      * **Finding a free + workable data source to enable near-real-time metrics for plus-minus analytics was NOT easy**. I ultimately found a functional solution with the “replay” feature at [SportsData.io](https://sportsdata.io/). Users who register for a free account have access to multiple “replays” which simulate historical periods of activity in a near-real-time update manner. For example, by starting the replay from December 2, 2023 via the SportsData.IO website, I received an API key for this particular replay and was able to obtain freshly updated data every 60 seconds. For data engineering bootcamp students wishing to test out a sports data pipeline but unwilling to pay for an API, this is the best thing imaginable short of sitting shotgun in the Delorean with Marty McFly.
      * The replay from [SportsData.io](https://sportsdata.io/) not only provides data for free, it gives users the flexibility to set the start time for their live data stream. This feature proved extremely useful for testing and validation.
  + Historical: [barttorvik.com](https://barttorvik.com/)
    - Discussion & justification:
      * Getting historical data for free was easier. But I still didn’t want to scrape it or pay for it. Enter Bart Torvik. For avid college basketball fans who are also avid data gurus, Bart is basically Taylor Swift. Except his eras are stored timestamp-partitioned databases…and I don’t have to hear about who he’s dating every time I try to watch a football game. Bart is amazing and well-known in the college hoops universe for his no frills, but amazing analytics. He discourages web scraping and other such nonsense by making most of his raw data available to curious quants for free. Within reason, he has a “just ask me - I’ll try to help” policy. So I did. And he did. Exchanging quick emails with Bart about logistics and schemas was definitely one of the highlights of this project for me.
      * Bart provided giant raw .json files which contained all player stats for all games in a given year. I chose to include such historical data starting in 2014.
* Tech Stack
  + Databricks
    - I chose a strategy of “do everything in Databricks” for this project. Why? I had a subscription as a result of joining the DataExpert.IO bootcamp. I chose to make this project my “learn Databricks by doing” vehicle.
  + Delta tables
    - Raw data was converted to json and/or csv, processed/transformed, and ultimately stored in delta tables. An append-able delta table was used for storing near-real-time data after each ~ 60 second refresh via the SportsData.IO replay API. Delta Live Tables were used for all other data storage, including the metrics that serve the downstream Databricks dashboard.
  + ETL/Orchestration
    - Databricks Workflows and Pipelines were used. The integrated nature of these tools made this process easy once I learned the mechanics. Many companies whine about the cost of Databricks, but I understand why some are willing to pay a premium for these features. The UI was generally friendly and the overall experience generally positive. The monitoring features and integration of data quality checks were especially convenient.
  + Python/SQL
    - All code is written in either python or SQL. These are not only my favorite languages; they are also the syntax of choice for working with Delta Lakes and data pipelines in Databricks.
  + Data Quality Checks
    - Databricks-integrated checks:
      * Multiple checks are included for each data source via Expectations. This makes it easy to deal with null values, etc. when writing rows to a Delta Live Table. The default setting for these checks is to “Warn” but users also have the option to drop corresponding rows or cause the job to fail:
      * 
      * 
    - Python in-line checks
      * Python scripts dealt with multiple issues in cleaning and ingesting the data. For example: When I studied the historical data from Bart Torvik, I observed that players could have multiple player IDs in a given season. I noted it there were a small, but significant number of players that switched teams during the season. Some even switched teams multiple times during a season! I put logic in place to ensure that data written to my delta tables properly accounted for these types of anomalies in the raw data.

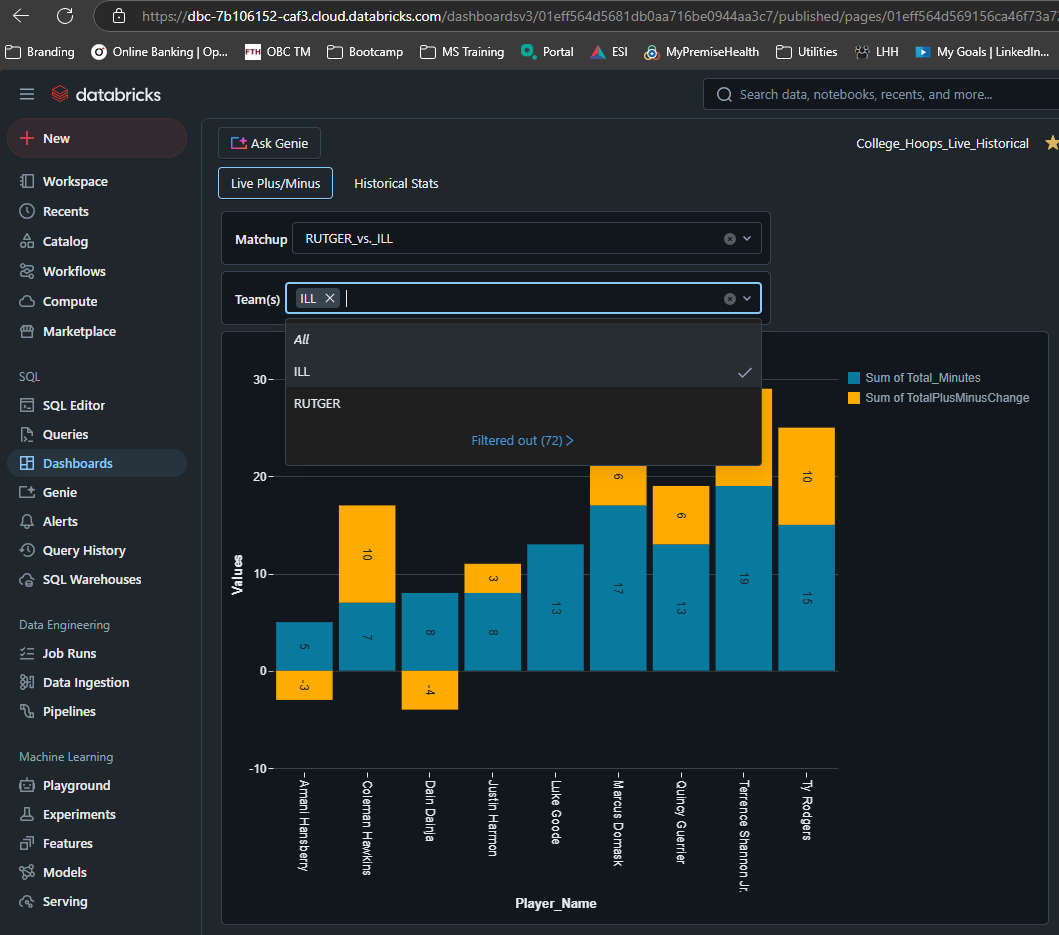
**Steps followed and challenges faced**

1. Get raw data. See previous section for challenges faced just to find workable data.
2. Study the data
   1. I spent 2 full days just studying the data from each source and learning the nuances that I would need to account for in my pipeline.
3. Clean/processing the raw data.
   1. I had to deal with nested JSON structures and other madness to achieve a workable dataset. There were redundant sections of data that needed to be excluded. Other sections that needed to be exploded or normalized before further downstream processing.
   2. I used a mixture of pandas DataFrames and Spark DataFrames. I know pandas is slow, but it’s what I know best. I chose a balance of pandas (to make use of my existing proficiency) and Spark (to force me to improve my PySpark skills).
4. Store the data
   1. I struggled to grasp the concept of Delta tables vs. Delta Live tables as it relates to appending data. Each time one of my Delta Live tables was refreshed, it would overwrite all of the pervious data. After several failed attempts to append and multiple consultations with fellow bootcamp students, I learned that this is the nature of Delta Live tables. I instead found a workaround solution for appending the results of my API updates to a non-live Delta table:

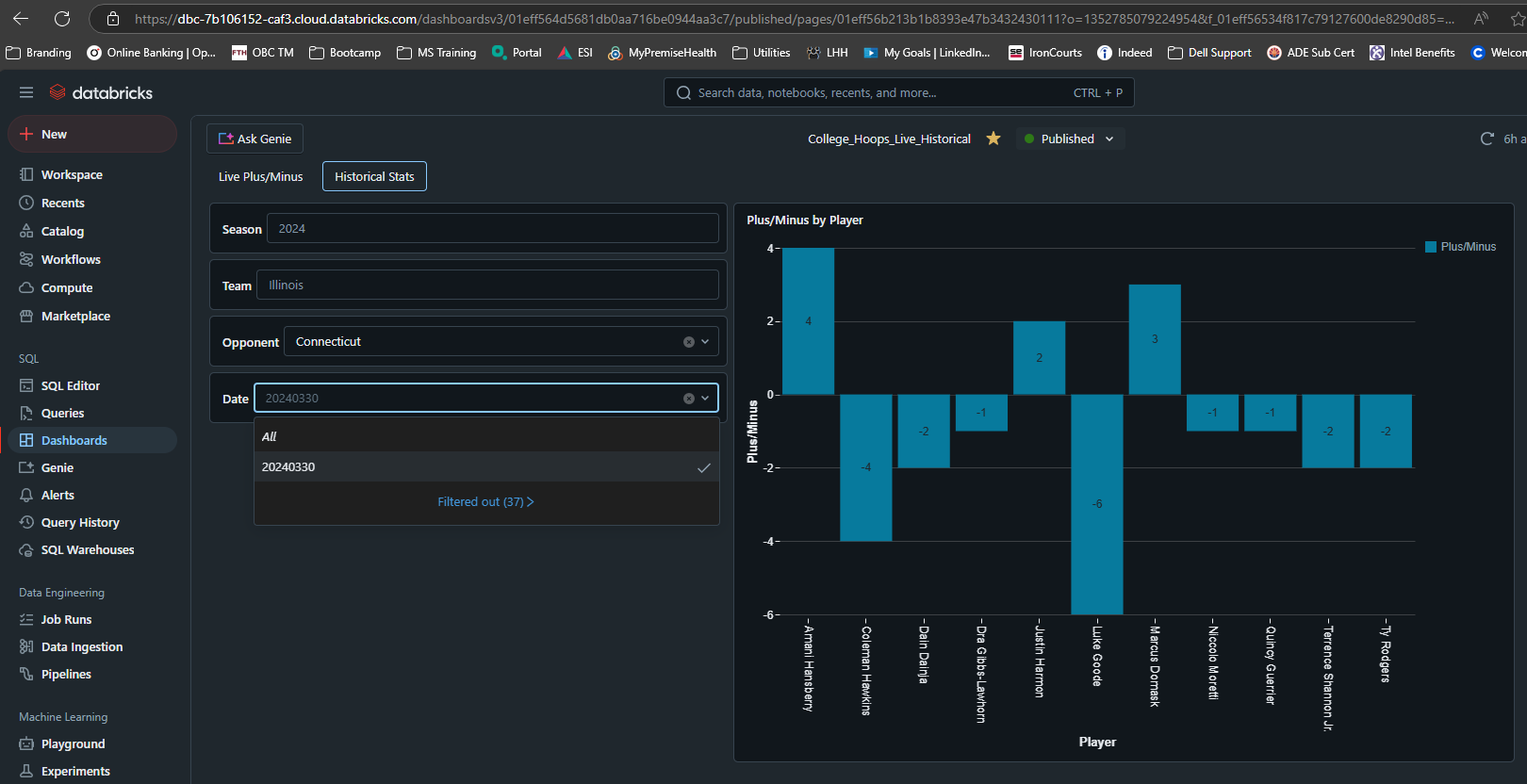


* 1. The crux of the plus-minus logic required that each I time I refreshed the live data, it was necessary to calculate the deltas since the previous updated. Specifically, I needed the change in minutes for each player AND the change in the score AND which players played for which team.
     1. The free data was not perfect. The live data feed occasionally showed a “bulk dump” behavior for less prominent games. For example, a player’s minute total might stay stagnant for 20 minutes. Then on the next update, the minute column value will change from 6 to 26. This abruption skews any downstream calculations because score changes corresponding to the 20-minute stagnation are lumped together and traceability is lost re: which players were on the court during each change in score. If such delays are 2 minutes instead of 1, that’s a rounding error. But 20 minutes of stagnation is not serviceable. My downstream metrics will only be as near-real-time accurate as my raw data API updates.

1. Create metrics from Delta tables
   1. This part was much easier. Once the data was stored strategically in a Delta Table, I only needed a little “lag magic” and some minimally painful group by statements to achieve the elusive plus-minus metric for each player in each game.
2. Create a Databricks Dashboard
   1. This was the easiest part of all. My ultimate visualization is simple and (to me) inspiring. It visually demonstrates (in near-real-time) the ratio of a player’s plus-minus metric to their total minutes played. **It makes it visually obvious when a coach is in love with a player that might be bringing the whole team closer to its ultimate demise with each passing minute!**
   2. The historical dashboard is less awe-inspiring, but still useful. Are you upset about how a coach is handling his team’s minutes while watching a live game? Click on the historical tab to see what happened when your team played this same opponent last year… or 5 years ago. “Those who cannot remember the past (or access it quickly via a side-by-side dashboard while monitoring live statistics) are condemned to repeat it.” Or something like that.

****

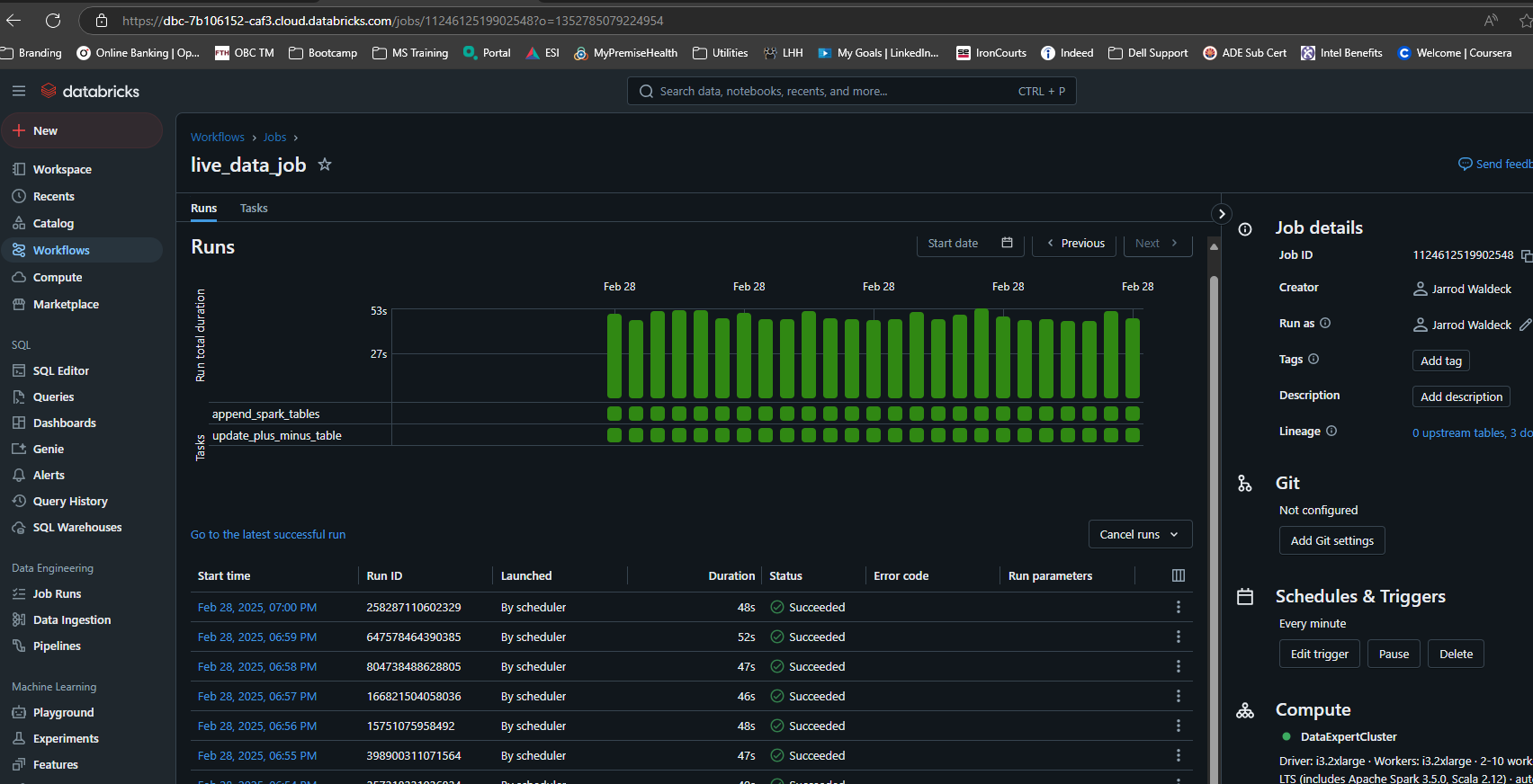
Screenshot of Live Dashboard



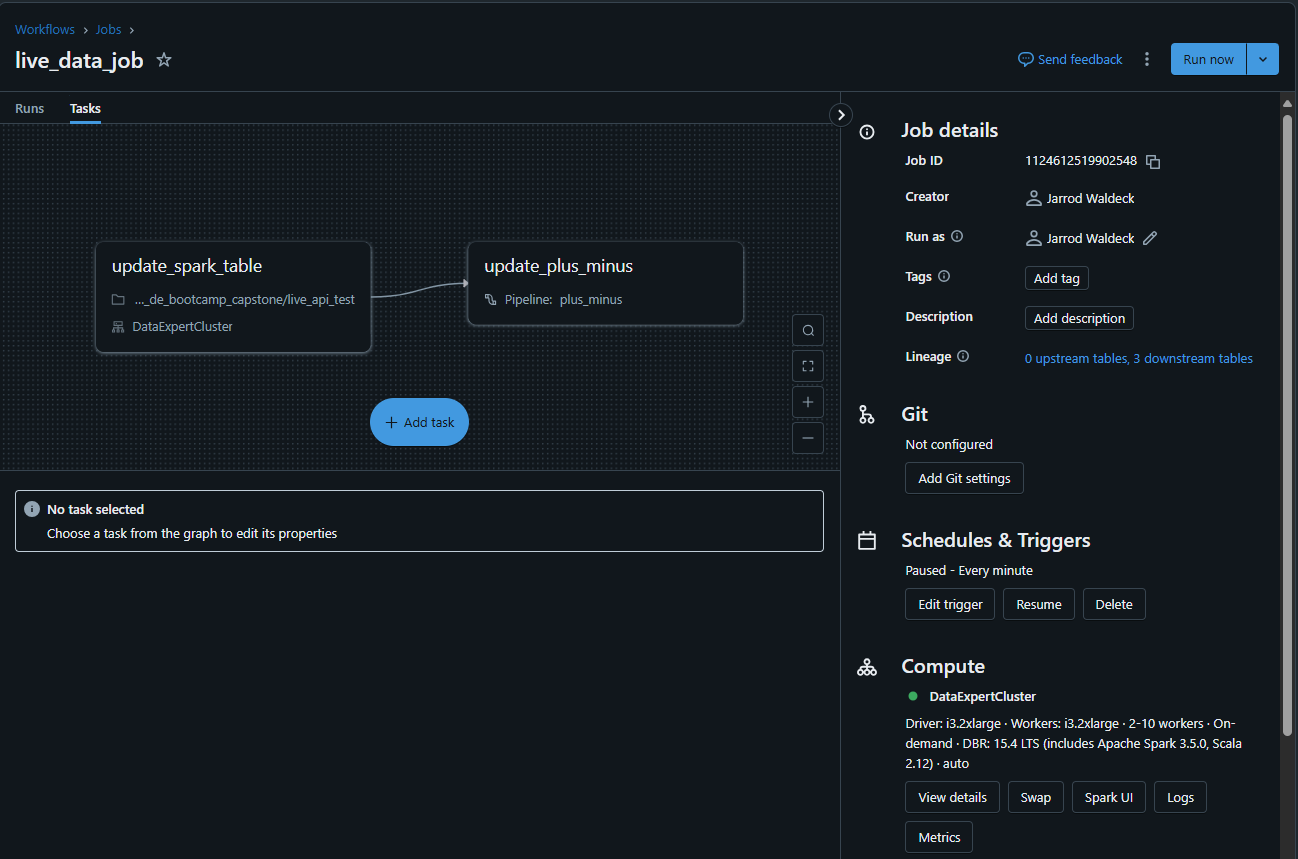
Screenshot of Historical Dashboard

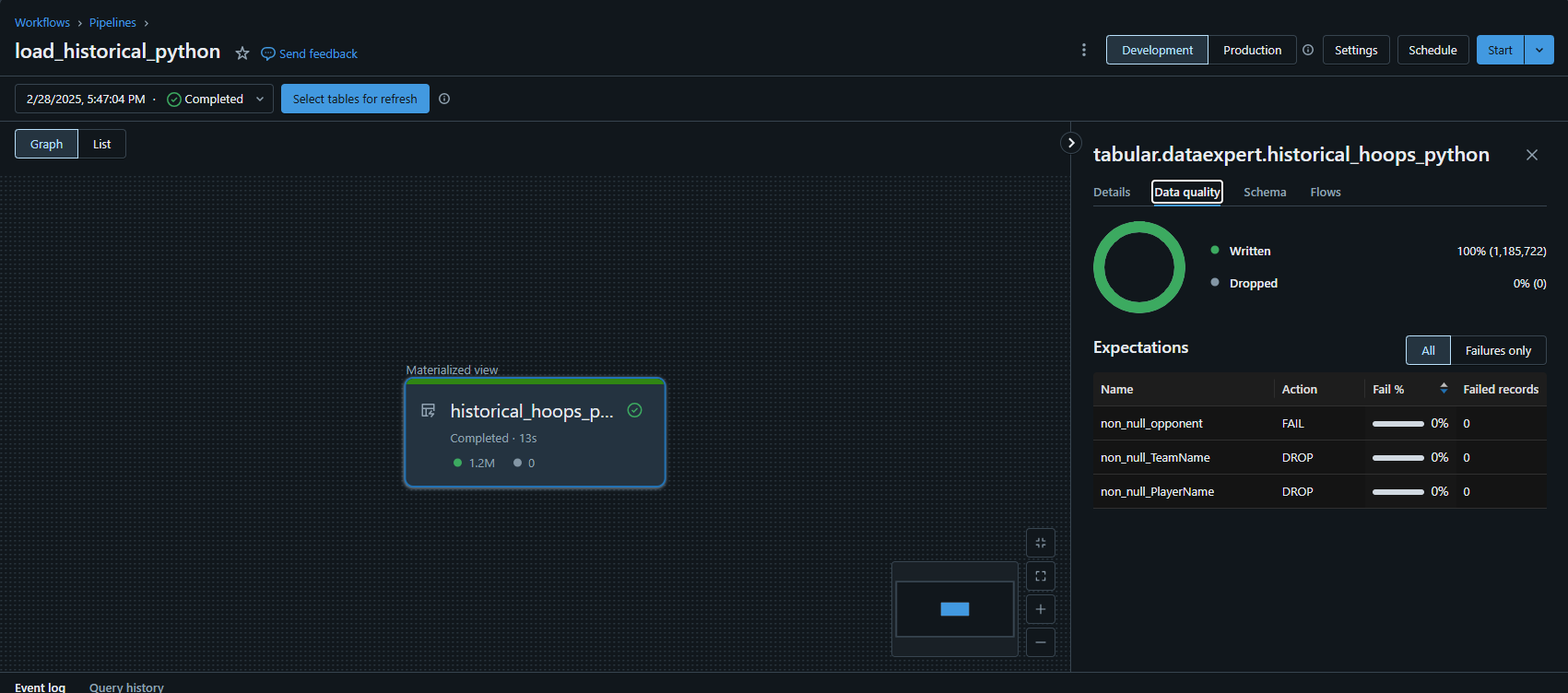
**Other Screenshots**

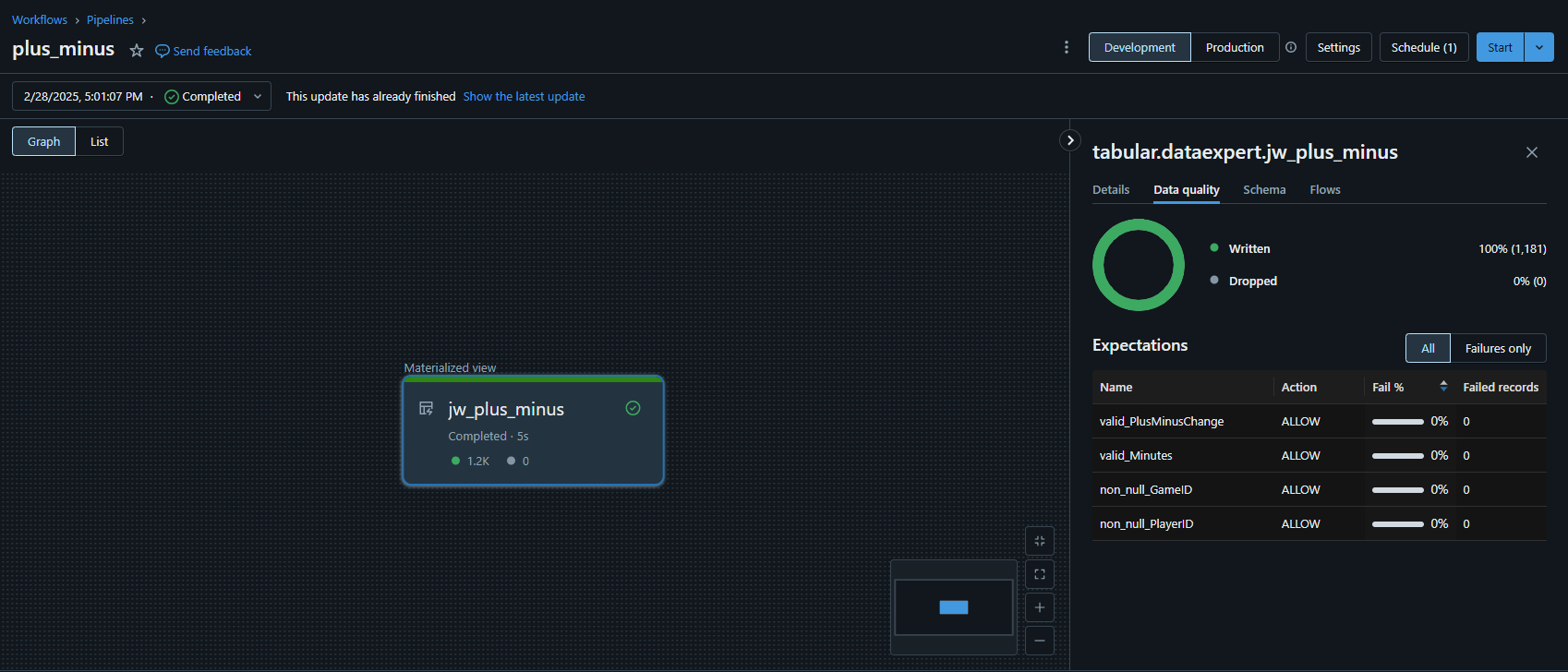
* **ETL runs**

****

* **DAG and Pipelines**



****

****

**TODOs + Possible future enhancements**

* Upgrade live data workflow conditions
  + When there are no games in progress, currently the 2nd task in the live update workflow fails. It doesn’t corrupt the data, but it makes the DAG graph look ugly. It would make more sense to somehow use the “Status” field in the raw games data to determine which tasks to run/skip. This would also save on cloud costs and would be a top priority for a true production solution.
* Add Stadium info from json
  + It was beyond the 2-week scope of this project, but the nested stadium info from the live updates includes latitude & longitude info. Might be run to include this as part of the Dashboard.
* Add/improve data quality checks
  + Given more time, I would add additional and more complex data quality checks. At minimum, I would move the checks that are buried in the python code and make them formally integrated Databricks Expectations.
* Summarize/archive live stats daily and load to historical
  + It would be nice to publish a game’s data (once it’s completed) to the historical archive. This way I could use Bart Torvik for previous seasons, but begin building my own historical database.
* Add more stats to the dashboard (e.g. final score of game)
  + Plenty of opportunity here. The data infrastructure is already built to add many more stats or custom metrics. I limited the scope strategically, but future is wide open with potential here.
* Productionize code processing the historical json data
  + This was done with a “one-time” mindset. In theory, I’d only get Bart Torvik data once per year going forward, so not a huge priority. Nevertheless, the script should still be functionalized and parameterized to ensure any future historical data updates are seamless.
* Move stuff from python code to config files (yaml, etc)
  + Move schema definitions to a separate config file
  + Move columns of interest specs (keep, drop, etc.) to separate config file
* Convert all pandas dfs to Spark dfs
  + Pandas sucks. I used it because it’s what I knew best. Now that I’m better at working with Spark DataFrames, I’d update the code to use those exclusively.
* Clean up README file
  + I copied and pasted some of this document into the repo’s README. I will store this file in the Docs folder, but the README needs a proper update so I can properly showcase this project.

**Data Volume:**

* Historical:
  + ~ 110K rows per year in each raw .json
  + historical\_hoops table populated with 11 years of history
    - i.e. 2014 – 2024 (inclusive)
  + 1.2M historical rows total
* Live:
  + games table:
    - 1 row per game per (update frequency) generated
      * Raw data source is updated every 1 min
      * Pipeline that updates game table also fires every 1 min
    - Daily data volume will depend on number of games on a given day
      * Example: Dec 2, 2023
        + 95 games \* 1 row per minute \* 1440 minutes per day = ~ 130K rows
  + player\_games table:
    - 1 row per player per game per (update frequency) generated
      * Raw data source is updated every 1 min
      * Pipeline that updates game table also fires every 1 min
    - Daily volume depends on:
      * Number of games in a given day
      * Number of players per team
      * The timing of the game
        + Player data doesn’t start returning via API until the relevant game starts.
        + Player data continues to return via API after game has completed within a given day
      * Example: Dec 2, 2023
        + Theoretical:

95 games \* 20 players per team \* 1 row per minute \* 1440 minutes per day = ~ 2.7M rows

Note: This is a worst-case scenario in which all games started at the beginning of the day.

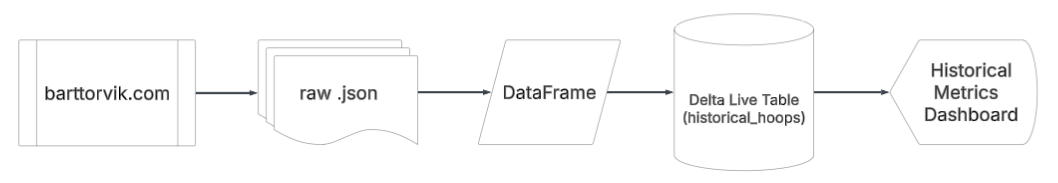
* + - * + Actual:

~ 1.2M live rows generated for this particular day

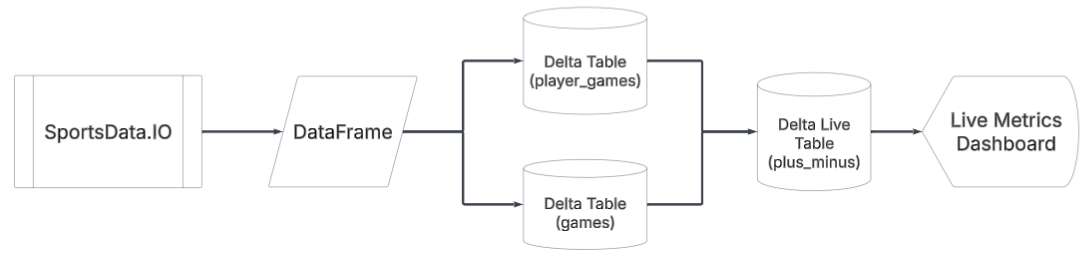
* + - * Future improvement note:
        + It should be possible to filter out rows to only include games that are “InProgress” when writing to the player\_games table. This would eliminate redundant rows for games that finish early in the day but continue to included in the API response for a given day.

**Data Flow:**

* Historical:



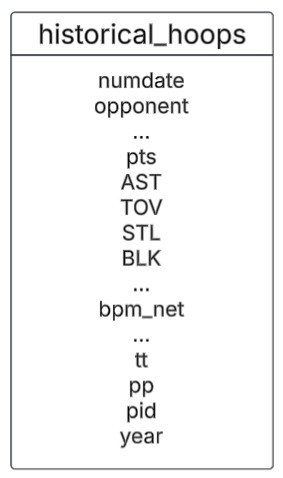
* Live:



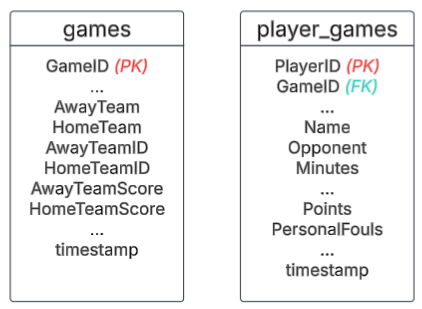
**Data Model**

**Note:** Primary key (PK) and foreign key (FK) designations should be interpreted as “*implicit and for documentation purposes only.*” Delta tables don’t enforce primary or foreign key constraints.

* Historical:



* Live:



**Delta Table Schema Info:**

* Historical:
  + raw source = *https://barttorvik.com/****yyyy****\_all\_advgames.json.gz*

|  |  |  |
| --- | --- | --- |
| **Table = historical\_hoops** | | |
| **Column** | **Type** | **Comments** |
| numdate | int | Numeric representation of game date (i.e. yyyymmdd) |
| win1 | int | Unknown, always 1 or 0. |
| opponent | string | Opponent team name (e.g. Texas). **Note: 351 unique values (teams) in 2018; 362 in 2024.** |
| win2 | int | Unknown. Always 1 or 0. |
| Min\_per | double | Unknown. Note: **58 unique values in 2018. Usually int-esque values between 1.0 - 58.0, but there's occasional 0.5 values.** |
| ORtg | double | Unknown. Possibly an offensive rating metric. |
| Usage | double | Estimate of the percentage of team plays a player was involved in while they were on the court. **Note: In 2023 values can exceed 300, which doesn't make sense.** |
| eFG | double | Unknown |
| TS\_per | double | Unknown |
| ORB\_per | double | Offensive rebound percentage. I.e. Percentage of available offensive rebounds a player grabs while they are on the court |
| DRB\_per | double | Defensive rebound percentage. I.e. Percentage of available defensive rebounds a player grabs while they are on the court |
| AST\_per | double | Assist percentage. i.e. Percentage of a player's teammates' field goals that the player assisted on while they were on the court. **Note: Has large negative values in 2018 which should be corrected to 0.** |
| TO\_per | double | Turnover percentage. I.e. Percentage of a player's possessions that end in a turnover. |
| dunksmade | int | Dunks made |
| dunksatt | int | Dunk attempted |
| rimmade | int | Shots made within close proximity (5 feet) of the hoop. |
| rimatt | int | Shots attempted within close proximity (5 feet) of the hoop. |
| midmade | int | Shots made within medium proximity (5 feet - 3pt range) of the hoop. |
| midatt | int | Shots attempted within medium proximity (5 feet - 3pt range) of the hoop. |
| twoPM | int | 2-point shots made in a game |
| twoPA | int | 2-point shots attempted in a game |
| TPM | int | 2-point shots made (redundant) in a game |
| TPA | int | 2-point shots attempted (redundant) in a game |
| FTM | int | Free throws made in a game |
| FTA | int | Free throws attempted in a game |
| bpm\_rd | double | Unknown. Some derivative of the plus/minus metric |
| Obpm | double | Unknown. Some derivative of the plus/minus metric |
| Dbpm | double | Unknown. Some derivative of the plus/minus metric |
| bpm\_net | double | Plus/Minus metric. I.e. Point differential when a player is on the court. **Note: This is the one Bart Torvik recommended using; closest to most commonly used plus/minus metric.** |
| pts | int | Points in a game |
| ORB | int | Offensive rebounds in a game |
| DRB | int | Defensive rebounds in a game |
| AST | int | Assists in a game |
| TOV | int | Turnovers in a game |
| STL | int | in a game |
| BLK | int | Blocks in a game |
| stl\_per | double | Steal percentage. I.e. Percentage of opponent possessions that end in a steal by a particular player while they are on the court |
| blk\_per | double | Block percentage. I.e. Percentage of opponent possessions that end in a block by a particular player while they are on the court |
| PF | int | Personal fouls in a game |
| possessions | int | Total possessions in a game |
| bpm | double | Unknown. Some derivative of the plus/minus metric |
| sbpm | double | Unknown. Some derivative of the plus/minus metric |
| tt | string | Team Name (e.g. Texas) |
| pp | string | Player Name (e.g. John Smith |
| inches | int | Height of player in inches |
| cls | string | Class of a player (e.g. Fr, So, Ju, Sr). **Note: 'NONE' exists in raw data for some years which should be correct to a blank string.** |
| pid | int | Player ID. **Note: In 2018 there are more unique player ids than unique player names. When players change teams during a season, they get a new pid.** |
| year | int | Year corresponding to when season ends. I.e. the 2022-2023 season will be 2023. |

* Live:
  + Raw source: *https://replay.sportsdata.io/api/v3/cbb/stats/...*

|  |  |  |
| --- | --- | --- |
| **Table = player\_games** | | |
| **Column** | **Type** | **Nullable** |
| StatID | int | TRUE |
| TeamID | int | TRUE |
| PlayerID ***(PK)*** | int | TRUE |
| SeasonType | int | TRUE |
| Season | int | TRUE |
| Name | string | TRUE |
| Team | string | TRUE |
| Position | string | TRUE |
| FanDuelSalary | int | TRUE |
| DraftKingsSalary | int | TRUE |
| FantasyDataSalary | int | TRUE |
| YahooSalary | int | TRUE |
| InjuryStatus | string | TRUE |
| InjuryBodyPart | string | TRUE |
| InjuryStartDate | timestamp | TRUE |
| InjuryNotes | string | TRUE |
| FanDuelPosition | string | TRUE |
| DraftKingsPosition | string | TRUE |
| YahooPosition | string | TRUE |
| OpponentRank | int | TRUE |
| OpponentPositionRank | int | TRUE |
| GlobalTeamID | int | TRUE |
| GameID ***(FK to games)*** | int | TRUE |
| OpponentID | int | TRUE |
| Opponent | string | TRUE |
| Day | timestamp | TRUE |
| DateTime | timestamp | TRUE |
| HomeOrAway | string | TRUE |
| IsGameOver | bool | TRUE |
| GlobalGameID | int | TRUE |
| GlobalOpponentID | int | TRUE |
| Updated | timestamp | TRUE |
| Games | int | TRUE |
| FantasyPoints | double | TRUE |
| Minutes | int | TRUE |
| FieldGoalsMade | int | TRUE |
| FieldGoalsAttempted | int | TRUE |
| FieldGoalsPercentage | double | TRUE |
| EffectiveFieldGoalsPercentage | double | TRUE |
| TwoPointersMade | int | TRUE |
| TwoPointersAttempted | int | TRUE |
| TwoPointersPercentage | double | TRUE |
| ThreePointersMade | int | TRUE |
| ThreePointersAttempted | int | TRUE |
| ThreePointersPercentage | double | TRUE |
| FreeThrowsMade | int | TRUE |
| FreeThrowsAttempted | int | TRUE |
| FreeThrowsPercentage | double | TRUE |
| OffensiveRebounds | int | TRUE |
| DefensiveRebounds | int | TRUE |
| Rebounds | int | TRUE |
| OffensiveReboundsPercentage | double | TRUE |
| DefensiveReboundsPercentage | double | TRUE |
| TotalReboundsPercentage | double | TRUE |
| Assists | int | TRUE |
| Steals | int | TRUE |
| BlockedShots | int | TRUE |
| Turnovers | int | TRUE |
| PersonalFouls | int | TRUE |
| Points | int | TRUE |
| TrueShootingAttempts | double | TRUE |
| TrueShootingPercentage | double | TRUE |
| PlayerEfficiencyRating | double | TRUE |
| AssistsPercentage | double | TRUE |
| StealsPercentage | double | TRUE |
| BlocksPercentage | double | TRUE |
| TurnOversPercentage | double | TRUE |
| UsageRatePercentage | double | TRUE |
| FantasyPointsFanDuel | double | TRUE |
| FantasyPointsDraftKings | double | TRUE |
| FantasyPointsYahoo | double | TRUE |
| Timestamp (FK) | timestamp | TRUE |

|  |  |  |
| --- | --- | --- |
| **Table = games** | | |
| **Column** | **Type** | **Nullable** |
| GameID ***(PK)*** | int | TRUE |
| Season | int | TRUE |
| SeasonType | int | TRUE |
| Status | string | TRUE |
| Day | timestamp | TRUE |
| DateTime | timestamp | TRUE |
| AwayTeam | string | TRUE |
| HomeTeam | string | TRUE |
| AwayTeamID | int | TRUE |
| HomeTeamID | int | TRUE |
| AwayTeamScore | int | TRUE |
| HomeTeamScore | int | TRUE |
| Updated | timestamp | TRUE |
| Period | string | TRUE |
| TimeRemainingMinutes | int | TRUE |
| TimeRemainingSeconds | int | TRUE |
| PointSpread | double | TRUE |
| OverUnder | double | TRUE |
| AwayTeamMoneyLine | int | TRUE |
| HomeTeamMoneyLine | int | TRUE |
| GlobalGameID | int | TRUE |
| GlobalAwayTeamID | int | TRUE |
| GlobalHomeTeamID | int | TRUE |
| TournamentID | string | TRUE |
| Bracket | string | TRUE |
| Round | string | TRUE |
| AwayTeamSeed | string | TRUE |
| HomeTeamSeed | string | TRUE |
| AwayTeamPreviousGameID | string | TRUE |
| HomeTeamPreviousGameID | string | TRUE |
| AwayTeamPreviousGlobalGameID | string | TRUE |
| HomeTeamPreviousGlobalGameID | string | TRUE |
| TournamentDisplayOrder | string | TRUE |
| TournamentDisplayOrderForHomeTeam | string | TRUE |
| IsClosed | bool | TRUE |
| GameEndDateTime | timestamp | TRUE |
| HomeRotationNumber | int | TRUE |
| AwayRotationNumber | int | TRUE |
| TopTeamPreviousGameId | string | TRUE |
| BottomTeamPreviousGameId | string | TRUE |
| Channel | string | TRUE |
| NeutralVenue | bool | TRUE |
| AwayPointSpreadPayout | int | TRUE |
| HomePointSpreadPayout | int | TRUE |
| OverPayout | int | TRUE |
| UnderPayout | int | TRUE |
| DateTimeUTC | y | TRUE |
| Attendance | string | TRUE |
| timestamp | timestamp | TRUE |