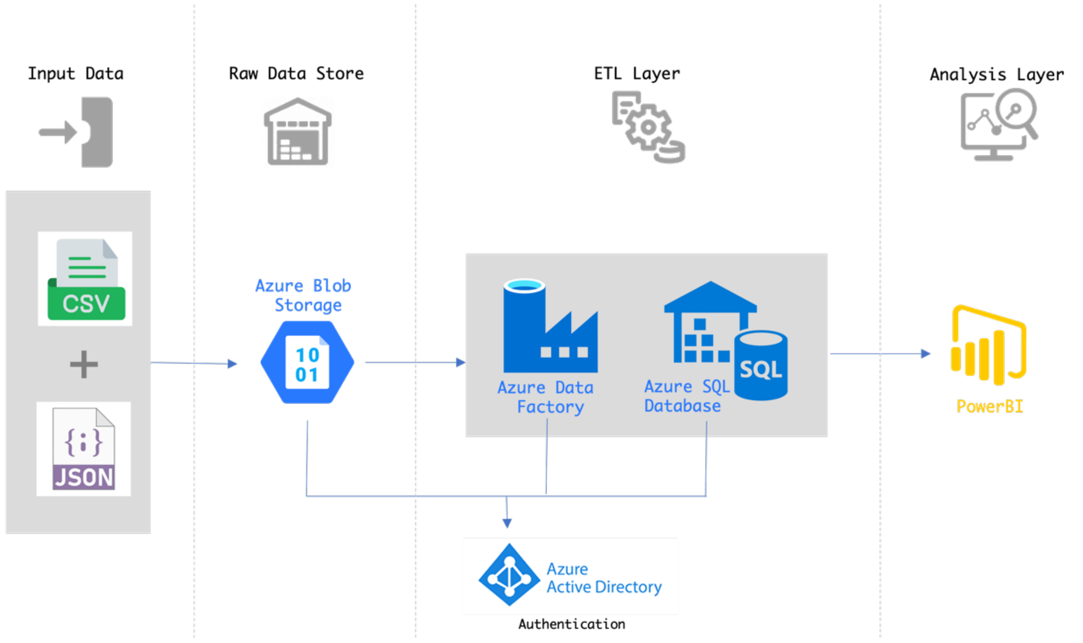
DAMG 7275 : Advanced Data Management Systems
Project P3 Submission

Topic : NBA Game Analytics
Team : Team 6 – Azure SQL Multi-model
Team Members :

Krishika Singh- 002194016
Mohammad Rafi Shaik- 001525707
Shika Shyam- 002194543
Shreya Soni – 002146758

Implementation Process :

Our Project architecture is as shown below :



- The data we are using is available in the NBA dataset provided by Kaggle. The data is in the form of multiple csv files for different entities.
- Since these CSV files are not normalized and has a lot of redundant columns we will be needing extensive preprocessing before we can use the data for reporting.

Our Implementation Process consists of three phases :

Step 1 : Raw Data Layer

- We are using Azure Blob Storage to store the csv files from Kaggle.
- We are also using Blob storage as the staging area for JSON documents.

Step 2 : ETL Layer

- Azure Data Factory is our ETL tool. We are using Data Copy activity for some entities and full-fledged data flows for others.
- We have also enabled both schedule based and storage event-based triggers.

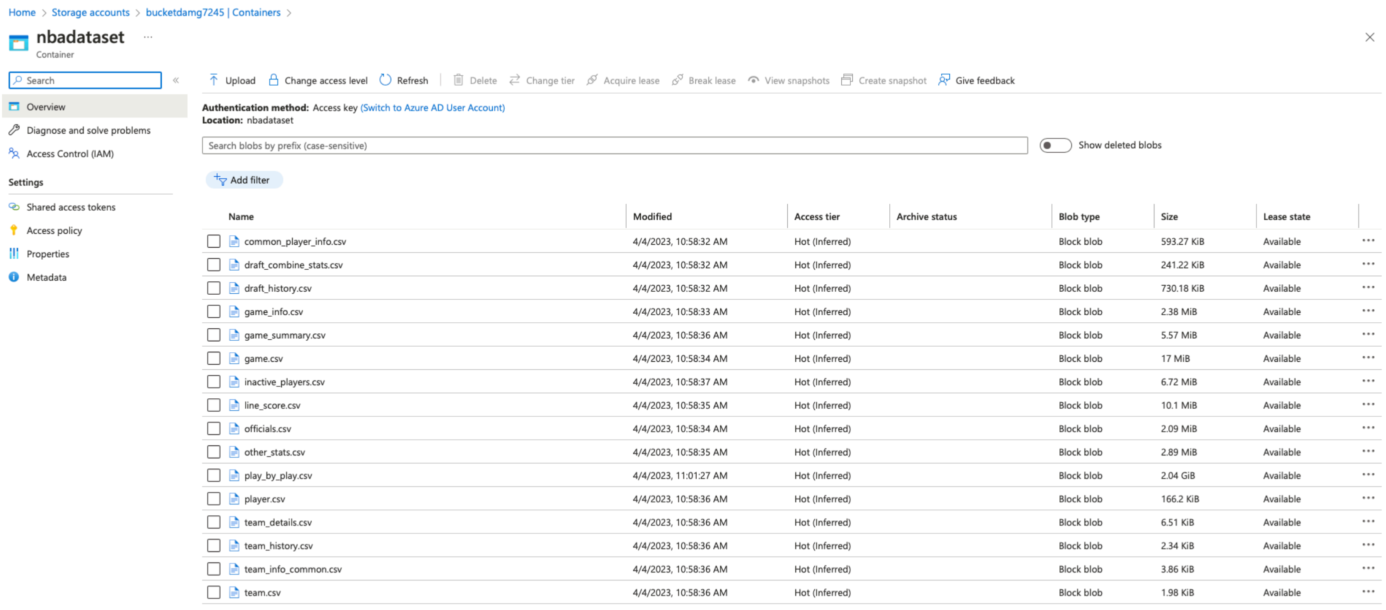
Step 3 : Foundation Data Layer

- The sink of our ETL pipelines point to Azure SQL tables.
- We are storing data as tables, nodes and edges for graphs as well as document.

Raw Data Store :

We have made use of Azure Blob Storage as shown below to store our CSVs. The source for all our ETLs will be pointing to this storage location.

nbadataset is the container where all our source csv files are available.



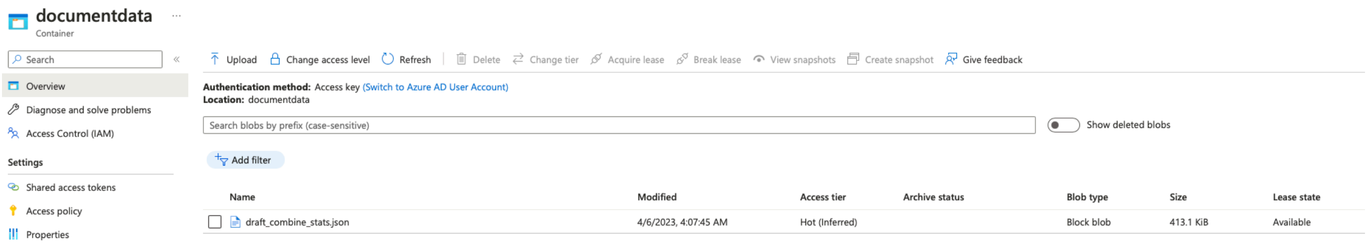
Home > Storage accounts > bucketdamg7245 | Containers >

nbadataset
Container

Authentication method: Access key (Switch to Azure AD User Account)
Location: nbadataset

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
common_player_info.csv	4/4/2023, 10:58:32 AM	Hot (Inferred)		Block blob	593.27 KiB	Available
draft_combine_stats.csv	4/4/2023, 10:58:32 AM	Hot (Inferred)		Block blob	241.22 KiB	Available
draft_history.csv	4/4/2023, 10:58:32 AM	Hot (Inferred)		Block blob	730.18 KiB	Available
game_info.csv	4/4/2023, 10:58:33 AM	Hot (Inferred)		Block blob	2.38 MiB	Available
game_summary.csv	4/4/2023, 10:58:36 AM	Hot (Inferred)		Block blob	5.57 MiB	Available
game.csv	4/4/2023, 10:58:34 AM	Hot (Inferred)		Block blob	17 MiB	Available
inactive_players.csv	4/4/2023, 10:58:37 AM	Hot (Inferred)		Block blob	6.72 MiB	Available
line_score.csv	4/4/2023, 10:58:35 AM	Hot (Inferred)		Block blob	10.1 MiB	Available
officials.csv	4/4/2023, 10:58:34 AM	Hot (Inferred)		Block blob	2.09 MiB	Available
other_stats.csv	4/4/2023, 10:58:35 AM	Hot (Inferred)		Block blob	2.89 MiB	Available
play_by_play.csv	4/4/2023, 11:01:27 AM	Hot (Inferred)		Block blob	2.04 GiB	Available
player.csv	4/4/2023, 10:58:36 AM	Hot (Inferred)		Block blob	166.2 KiB	Available
team_details.csv	4/4/2023, 10:58:36 AM	Hot (Inferred)		Block blob	6.51 KiB	Available
team_history.csv	4/4/2023, 10:58:36 AM	Hot (Inferred)		Block blob	2.34 KiB	Available
team_info_common.csv	4/4/2023, 10:58:36 AM	Hot (Inferred)		Block blob	3.86 KiB	Available
team.csv	4/4/2023, 10:58:36 AM	Hot (Inferred)		Block blob	1.98 KiB	Available

Additionally, we use documentdata container to store the staging json file that we create from the csv in order to store as document in foundation layer.



documentdata
Container

Authentication method: Access key (Switch to Azure AD User Account)
Location: documentdata

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
draft_combine_stats.json	4/6/2023, 4:07:45 AM	Hot (Inferred)		Block blob	413.1 KiB	Available

ETL Layer:

Since we have multiple source csv files our first step in ETL Design was to create a **consolidated data dictionary** as shown below where we determine from which file the source data for each target column in a given table comes from.

Microsoft Excel Worksheet

We created this for all columns in every single table in our Foundation layer.

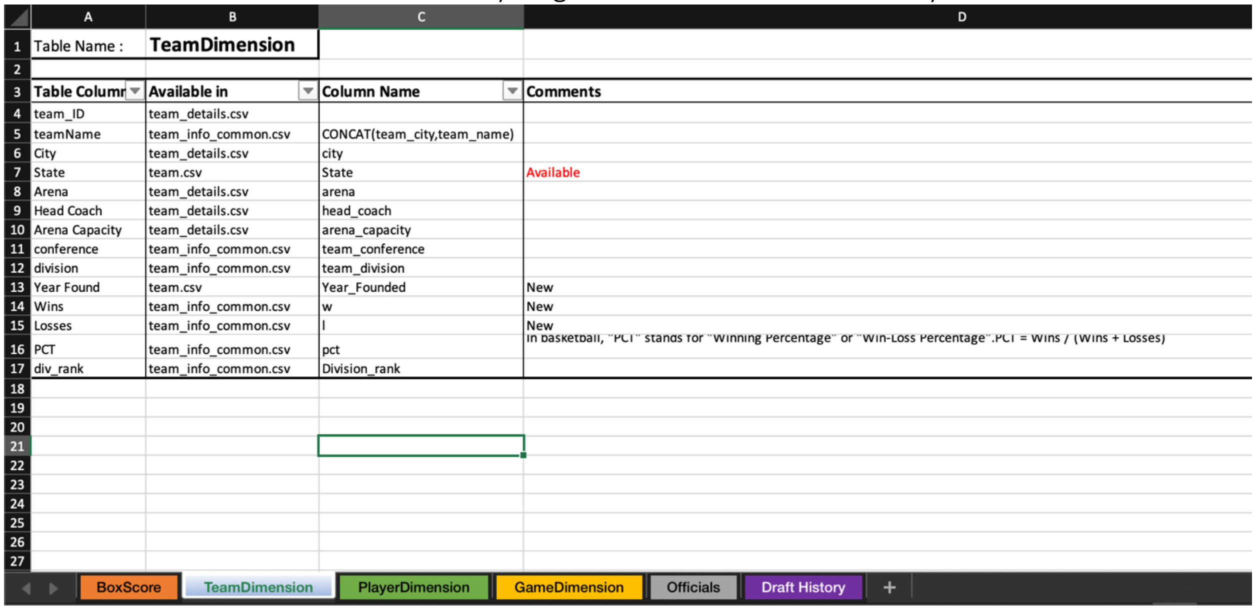
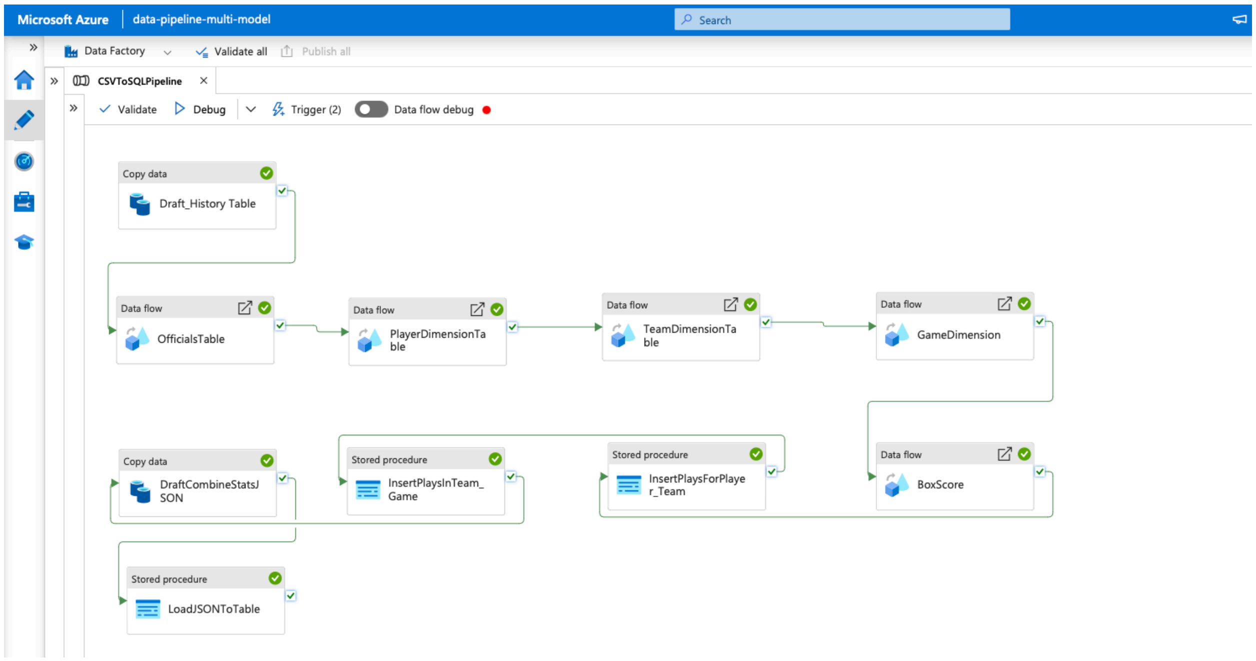


Table Name :	TeamDimension		
Table Column	Available in	Column Name	Comments
team_ID	team_details.csv		
teamName	team_info_common.csv	CONCAT(team_city,team_name)	
City	team_details.csv	city	
State	team.csv	State	Available
Arena	team_details.csv	arena	
Head Coach	team_details.csv	head_coach	
Arena Capacity	team_details.csv	arena_capacity	
conference	team_info_common.csv	team_conference	
division	team_info_common.csv	team_division	
Year Found	team.csv	Year_Founded	New
Wins	team_info_common.csv	w	New
Losses	team_info_common.csv	l	New
PCT	team_info_common.csv	pct	In basketball, "PCT" stands for "Winning Percentage" or "Win-Loss Percentage".PCT = Wins / (Wins + Losses)
div_rank	team_info_common.csv	Division_rank	

ETL Pipeline in Azure Data Factory:

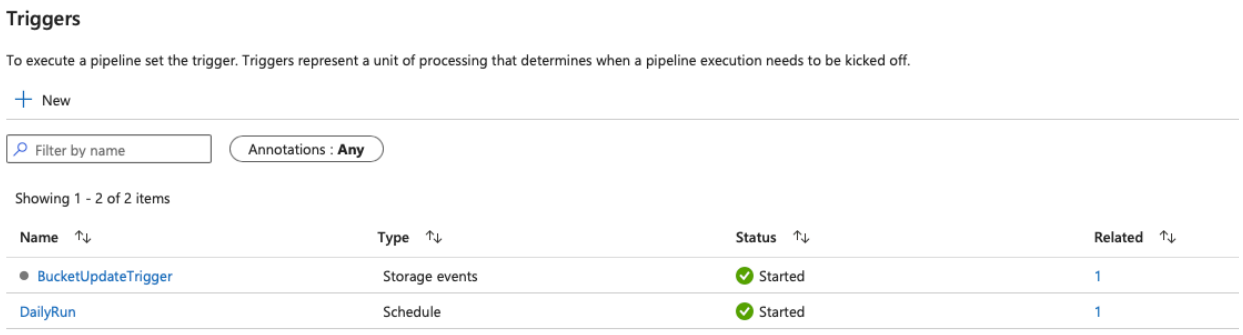
We have created the below ETL pipeline for our project. It loads each entity in the order of execution, and runs the required stored procedures in target after loading.
Due to the cost constraints that come with using heavy clusters for ADF pipeline executions, we have traded off on the parallel runs and decided to run each component of the ETL sequentially. In an ideal scenario with unlimited costs and unlimited compute resources available – we could possibly run most of our pipeline activities in parallel.
Another reason to run the jobs sequentially is due to the foreign key constraints between tables that require data to be available in some tables before it Is available in other tables.



Pipeline execution triggers :

We have set up two types of triggers for our pipeline –

- There is a Scheduled trigger called Daily Run – which is scheduled to run everyday at 12AM UTC
- There is a Storage Events Trigger called BucketUpdateTrigger- which will run every time there is a new file added or updated in our Raw Data Storage Container



Triggers

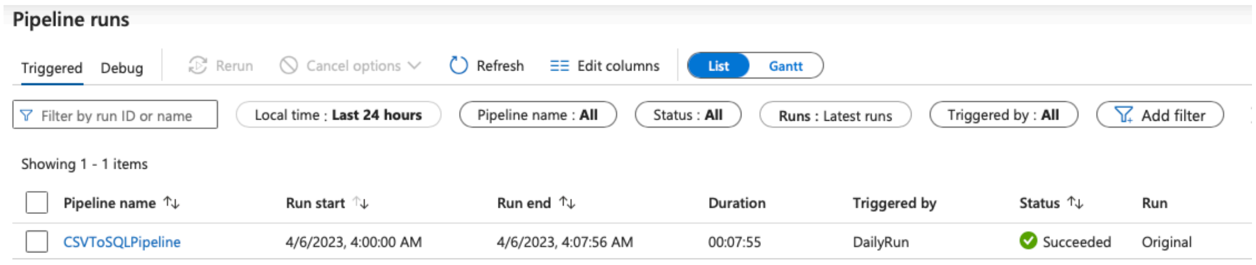
To execute a pipeline set the trigger. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off.

+ New

Filter by name | Annotations : **Any**

Showing 1 - 2 of 2 items

Name	Type	Status	Related
● BucketUpdateTrigger	Storage events	✓ Started	1
DailyRun	Schedule	✓ Started	1



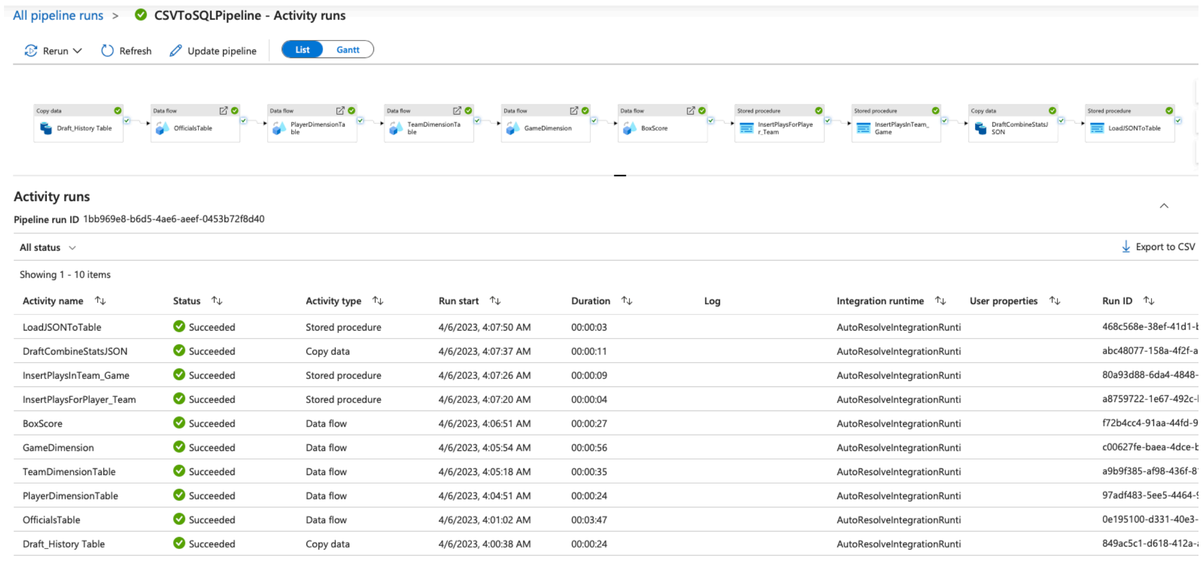
Pipeline runs

Triggered | Debug | Rerun | Cancel options | Refresh | Edit columns | List | Gantt

Filter by run ID or name | Local time : **Last 24 hours** | Pipeline name : **All** | Status : **All** | Runs : Latest runs | Triggered by : **All** | Add filter

Showing 1 - 1 items

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run
CSVToSQLPipeline	4/6/2023, 4:00:00 AM	4/6/2023, 4:07:56 AM	00:07:55	DailyRun	✓ Succeeded	Original


All pipeline runs > CSVToSQLPipeline - Activity runs

Activity runs

Pipeline run ID 1bb969e8-b6d5-4ae6-aeef-0453b72f8d40

All status

Export to CSV

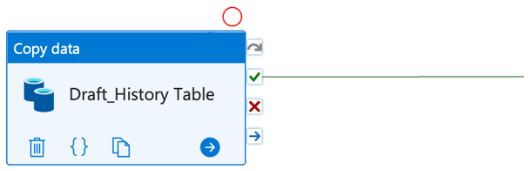
Showing 1 - 10 items

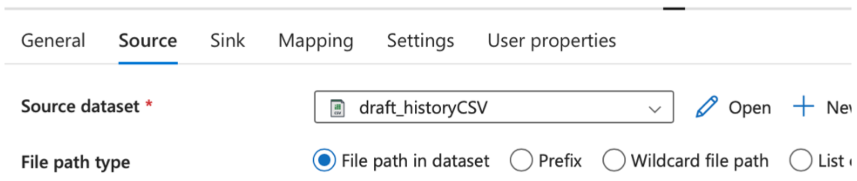
Activity name	Status	Activity type	Run start	Duration	Log	Integration runtime	User properties	Run ID
LoadJSONToTable	Succeeded	Stored procedure	4/6/2023, 4:07:50 AM	00:00:03		AutoResolveIntegrationRunti		468c568e-38ef-41d1-
DraftCombineStatsJSON	Succeeded	Copy data	4/6/2023, 4:07:37 AM	00:00:11		AutoResolveIntegrationRunti		abc48077-158a-4f2f-a
InsertPlaysInTeam_Game	Succeeded	Stored procedure	4/6/2023, 4:07:26 AM	00:00:09		AutoResolveIntegrationRunti		80a93d88-6da4-4848-
InsertPlaysForPlayer_Team	Succeeded	Stored procedure	4/6/2023, 4:07:20 AM	00:00:04		AutoResolveIntegrationRunti		a8759722-1e67-492c-
BoxScore	Succeeded	Data flow	4/6/2023, 4:06:51 AM	00:00:27		AutoResolveIntegrationRunti		f72b4cc4-91aa-44fd-9
GameDimension	Succeeded	Data flow	4/6/2023, 4:05:54 AM	00:00:56		AutoResolveIntegrationRunti		c00627fe-baea-4dce-
TeamDimensionTable	Succeeded	Data flow	4/6/2023, 4:05:18 AM	00:00:35		AutoResolveIntegrationRunti		a9b9f385-af98-436f-8
PlayerDimensionTable	Succeeded	Data flow	4/6/2023, 4:04:51 AM	00:00:24		AutoResolveIntegrationRunti		97adf483-5ee5-4464-
OfficialsTable	Succeeded	Data flow	4/6/2023, 4:01:02 AM	00:03:47		AutoResolveIntegrationRunti		0e195100-d331-40e3-
Draft_History Table	Succeeded	Copy data	4/6/2023, 4:00:38 AM	00:00:24		AutoResolveIntegrationRunti		849ac5c1-d618-412a-

The above image shows all steps of our ADF pipeline has run and completed successfully.

Our pipeline consists of three major types of ADF activities :

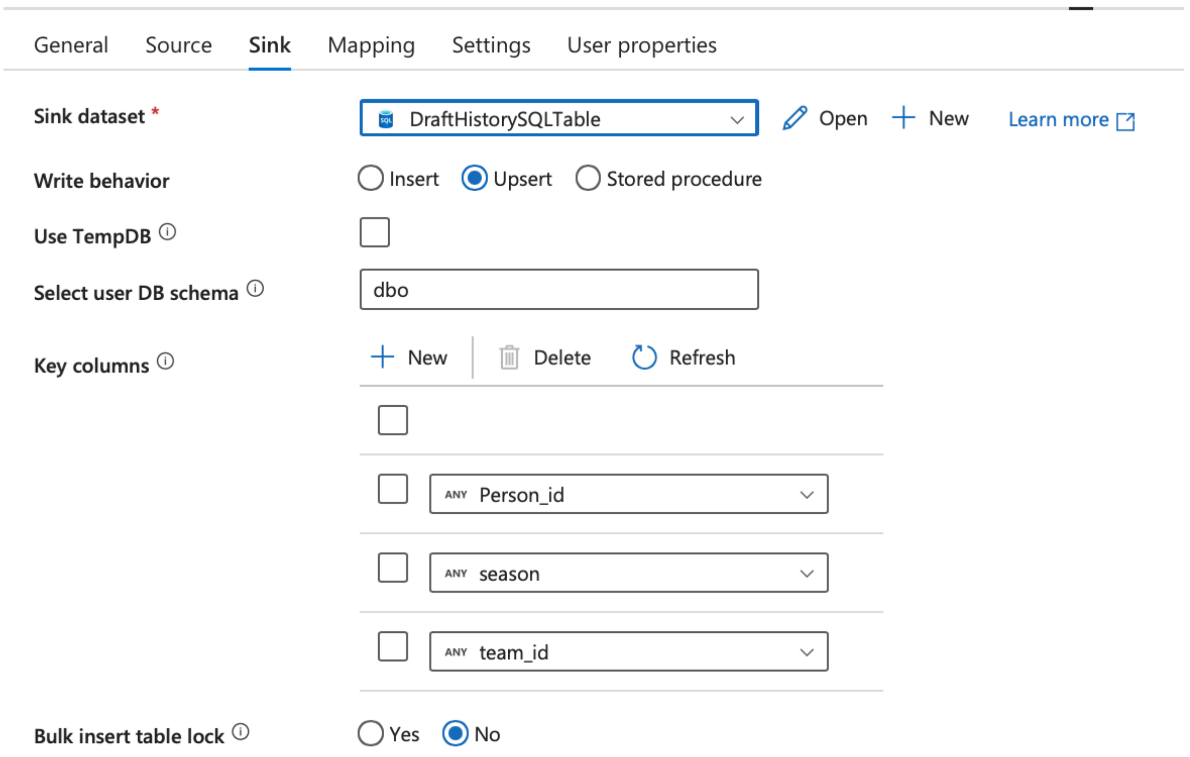
1. **DataCopy Activity :**
   i. We use Data Copy activity to populate DraftHistory table : Since this table requires a direct pull with the omission of a few columns from one single source csv file, we can leverage Copy Data Activity Here.





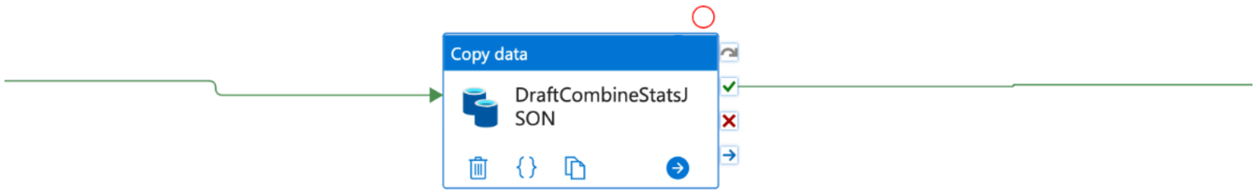
Source dataset * draft_historyCSV

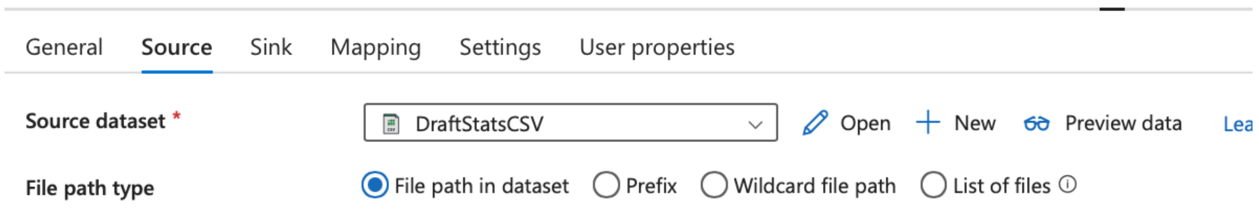
File path type [✓ File path in dataset] Prefix Wildcard file path List


General Source **Sink** Mapping Settings User properties

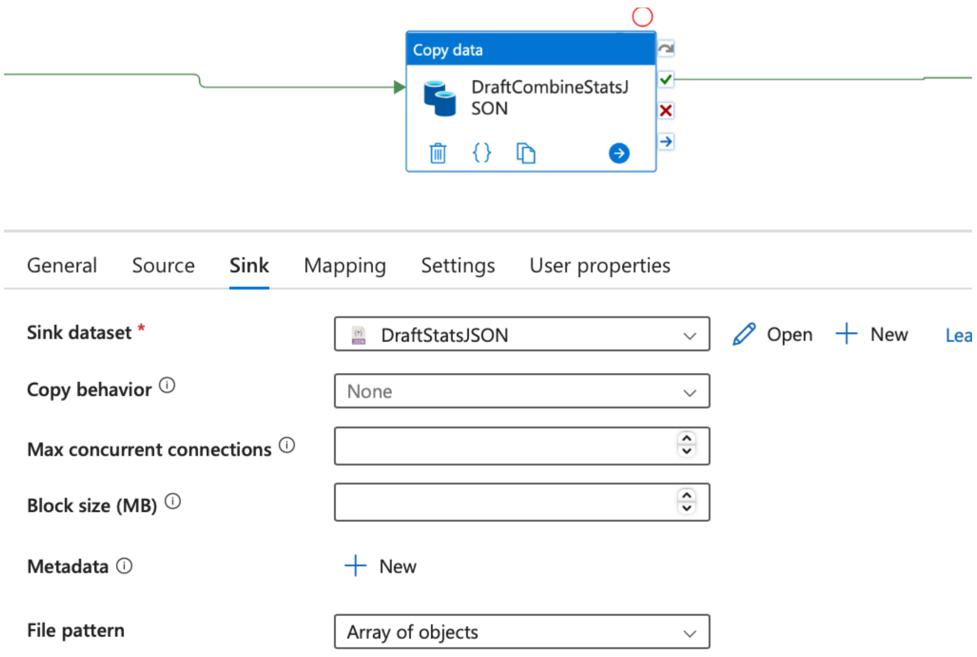
Sink dataset *	DraftHistorySQLTable
Write behavior	Insert [✓ Upsert] Stored procedure
Use TempDB	☐
Select user DB schema	dbo
Key columns	ANY Person_id; ANY season; ANY team_id
Bulk insert table lock	Yes [✓ No]

ii. We also use CopyData activity for converting the csv file into JSON format for our DocumentStore table – PlayerDraftStatistics




General **Source** Sink Mapping Settings User properties

Source dataset *	DraftStatsCSV
File path type	[✓ File path in dataset] Prefix Wildcard file path List of files



Sink dataset * DraftStatsJSON

Copy behavior None

Max concurrent connections

Block size (MB)

Metadata + New

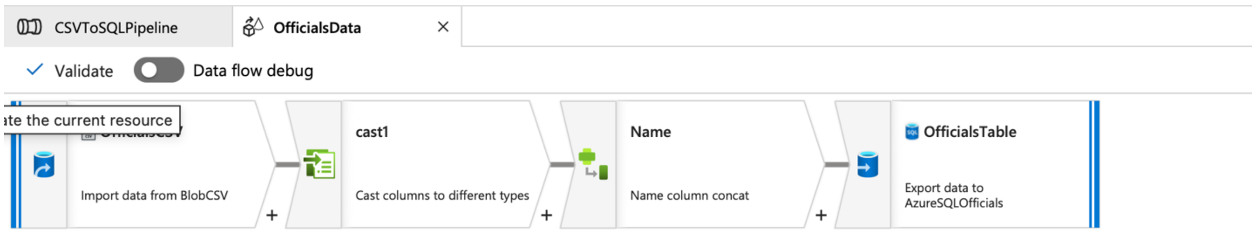
File pattern Array of objects

2. Data Flow Activity:

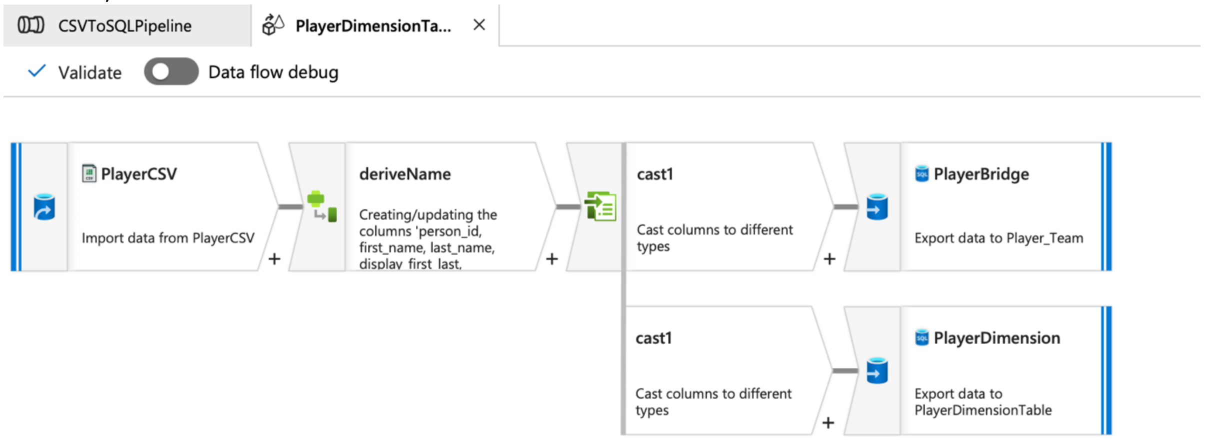
We have ADF Dataflow components for OfficialsTable PlayerDimensionTable, TeamDimension ,GameDimension and BoxScore.

We have made use of various mapping components available within AWS such as – Cast, Derived Columns, Lookups, Filters, Select, Aggregation component etc.

i. Officials Table:

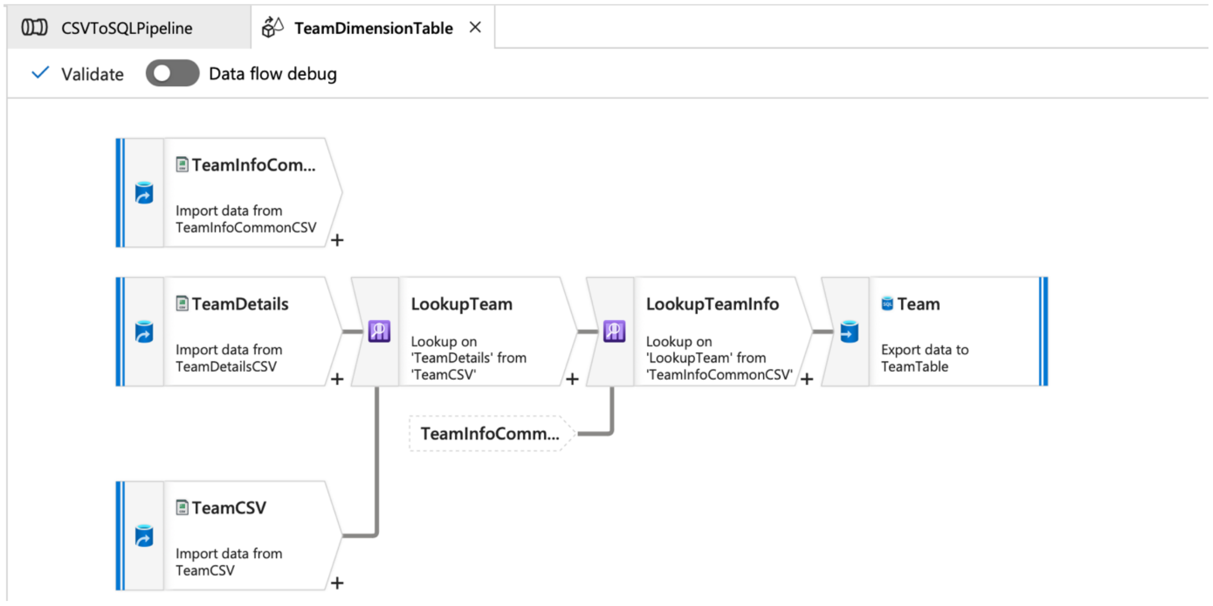


ii. Player Table :

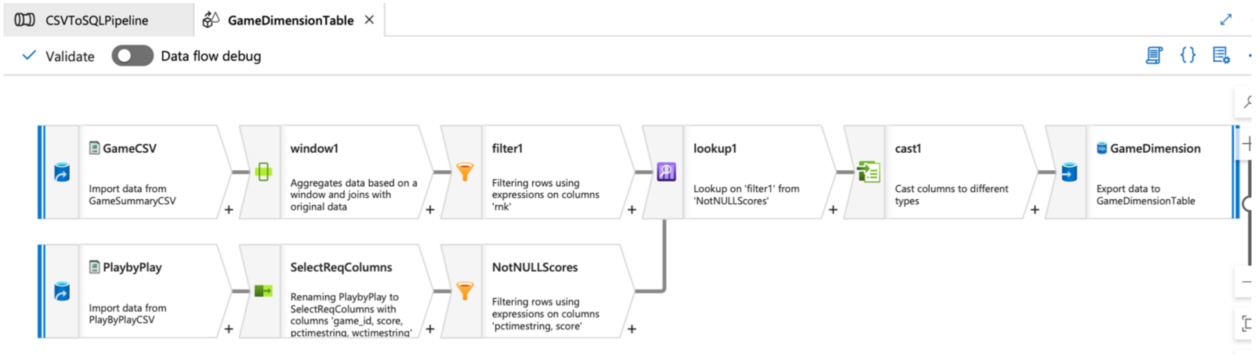

CSVToSQLPipeline
PlayerDimensionTa...
Validate
Data flow debug
PlayerCSV
Import data from PlayerCSV
deriveName
Creating/updating the columns 'person_id, first_name, last_name, display_first_last,
cast1
Cast columns to different types
PlayerBridge
Export data to Player_Team
cast1
Cast columns to different types
PlayerDimension
Export data to PlayerDimensionTable

In the above Data Flow we are creating two sink datasets – PlayerDimension table and PlayerBridge Table. PlayerBridge table acts as an associative entity to link players to teams – we use this same information in order to define edges between Player and Team in our Graph model.

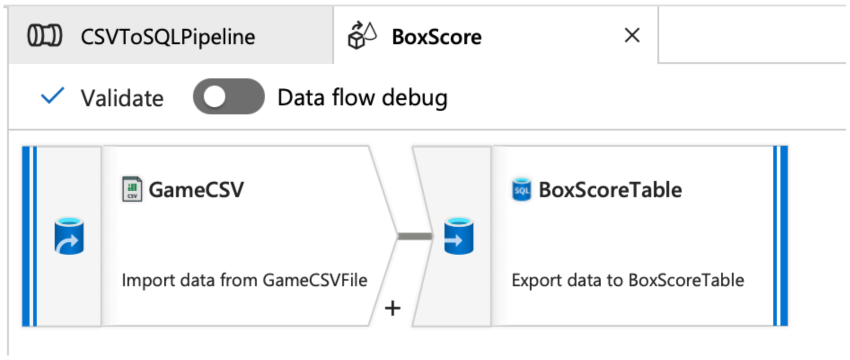
iii. TeamDimension Table :



iv. GameDimension Table:


CSVToSQLPipeline
GameDimensionTable
Validate
Data flow debug
GameCSV
Import data from GameSummaryCSV
window1
Aggregates data based on a window and joins with original data
filter1
Filtering rows using expressions on columns 'rnk'
lookup1
Lookup on 'filter1' from 'NOTNULLScores'
cast1
Cast columns to different types
GameDimension
Export data to GameDimensionTable
PlaybyPlay
Import data from PlayByPlayCSV
SelectReqColumns
Renaming PlaybyPlay to SelectReqColumns with columns 'game_id, score, pctimestring, wctimestring'
NotNULLScores
Filtering rows using expressions on columns 'pctimestring, score'

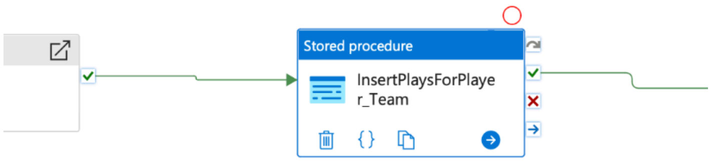
v. BoxScore Table :

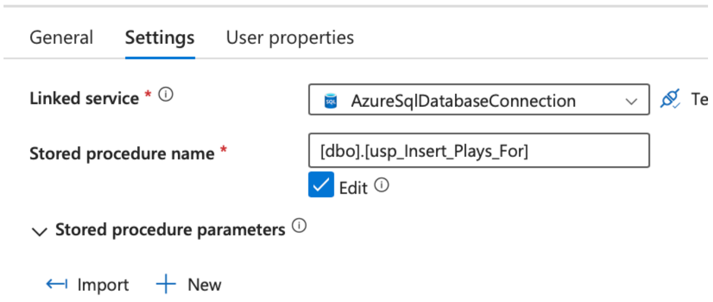


3. Stored Procedure Execution:

We use the Stored Procedure Execution component for executing three different stored procedures. We will explain the purpose of each Stored Proc later in this document :

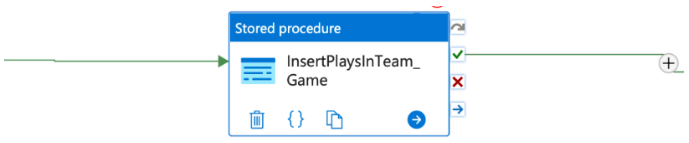
i. Usp_insert_plays_for Stored Proc

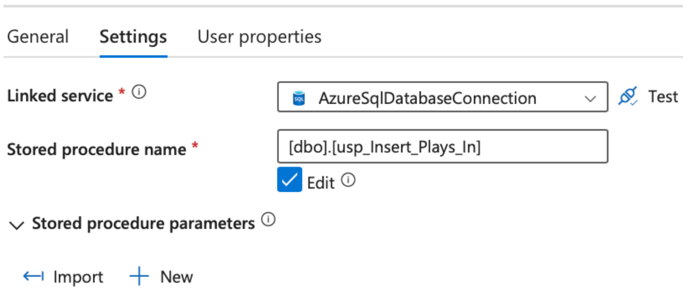



General | Settings | User properties

Linked service * : AzureSqlDatabaseConnection
Stored procedure name * : [dbo].[usp_Insert_Plays_For]
✓ Edit
Stored procedure parameters
Import | New

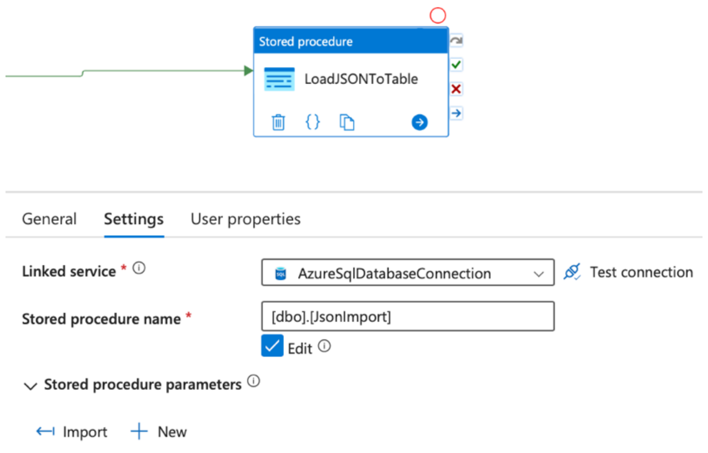
ii. Usp_insert_plays_in Stored Proc




General | Settings | User properties

Linked service * : AzureSqlDatabaseConnection
Stored procedure name * : [dbo].[usp_Insert_Plays_In]
✓ Edit
Stored procedure parameters
Import | New

iii. JsonImport Stored Proc


Stored procedure

LoadJSONToTable

General | Settings | User properties

Linked service * : AzureSqlDatabaseConnection — Test connection

Stored procedure name * : [dbo].[JsonImport]

✓ Edit

Stored procedure parameters

Import | New

Foundation Data Layer

Like described in the ERD, we have three types of Data Store within Azure SQL:

- Relational Data : Draft_History, Officials, BoxScore
- Document Data : PlayerDraftStatistics
- Graph Data : PlayerDimension, TeamDimension, GameDimension

The Relational Tables are created with the below DDLs and are loaded via ETL through Azure Data Factory as described above.

```
create table BoxScore
(
    Game_id                              int not null primary key,
    team_id_home                         int,
    team_id_away                         int,
    Minutes_played                       int,
    Points_hometeam                      int,
    Points_away                          int,
    field_goal_attempts_home             int,
    field_goal_Made_home                 int,
    field_goal_attempts_away             int,
    field_goal_Made_away                 int,
    three_point_field_goal_attempts_home int,
    three_point_field_goal_attempts_away int,
    three_point_field_goal_made_home     int,
    three_point_field_goal_made_away     int,
    free_throw_attempts_Home             int,
    free_throw_attempts_away             int,
    free_throw_made_Home                 int,
    free_throw_made_away                 int,
    reb_away                             int,
    reb_home                             int,
    assist_home                         int,
    assist_away                         int,
    Steal_home                          int,
    Steal_away                          int,
    Pf_away                             int,
    Pf_home                             int,
    Blocked_shots_away                  int,
    Blocked_shots_home                  int
)

create table Draft_history
(
    Person_id    int not null,
    Player_name  varchar(50),
    season       int not null,
    Round_number int,
    round_pick   int,
    team_id      int not null,
    primary key (Person_id, season, team_id)
)

create table Official
(
    Official_id int not null,
    Name        varchar(50),
    Game_id     int not null,
    Jerseynum   int,
    primary key (Official_id, Game_id)
)
```

The Document model is created with the below DDL, and loaded using the stored proc below that pulls the JSON data from the Azure Blob Storage which is defined as a DATA SOURCE within our nba SQL Server

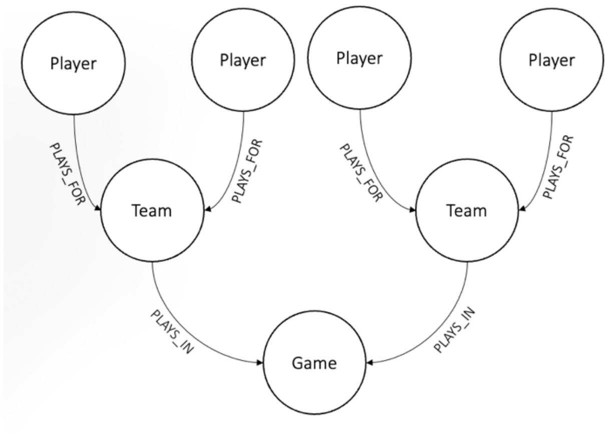
```
create table PlayerDraftStatistics
(
    JsonData nvarchar(max)
)
```

The below stored proc loads the PlayerDraftStatistics Document table. The Stored Proc is executed through the ADF pipeline.

```
CREATE PROCEDURE JsonImport
AS
BEGIN

TRUNCATE TABLE [dbo].[PlayerDraftStatistics]
BULK INSERT [dbo].[PlayerDraftStatistics]
```

```
FROM 'documentdata/draft_combine_stats.json'
WITH ( DATA_SOURCE = 'MyAzureBlobStorage');

END
```

The Graph Data Model is created using 3 different entities as shown in the Graph ERD- PlayerDimension, TeamDimension and GameDimension. The DDLs are below:



```
create table GameDimention
(
    Game_ID                                      int not null primary key,
    Date                                         date,
    home_team_id                                 int,
    away_team_id                                 int,
    Season                                       varchar(50),
    Score                                        varchar(50)
) AS NODE


create table Player_Dimension
(
    Player_id                                    int not null primary key,
    PlayerName                                   varchar(50),
    Position                                     varchar(50),
    Height                                       varchar(10),
    Weight                                       int,
    Birthdate                                    date,
    College                                      varchar(50),
    Country                                      varchar(50),
    season_experience                            int,
    Jersey                                       int,
    from_year                                    int,
    to_year                                      int,
    nba_flag                                     varchar(5)
) AS NODE
```

```
create table Team
(
    TeamID                                          int not null primary key,
    TeamName                                        varchar(255),
    City                                            varchar(255),
    State                                           varchar(255),
    Arena                                           varchar(255),
    ArenaCapacity                                   int,
    HeadCoach                                       varchar(255),
    Conference                                      varchar(255),
    TeamDivision                                    varchar(255),
    YearFounded                                     int,
    Wins                                            int,
    Losses                                          int,
    PCT                                             decimal(5, 3),
    Division_Rank                                   int
) AS NODE

create table Player_Team_Bridge
(
    Player_id int,
    Team_id   int
)
```

The edge tables are also defined using the below DDL and two stored procs are leveraged to define the relationship between the nodes based on the source data

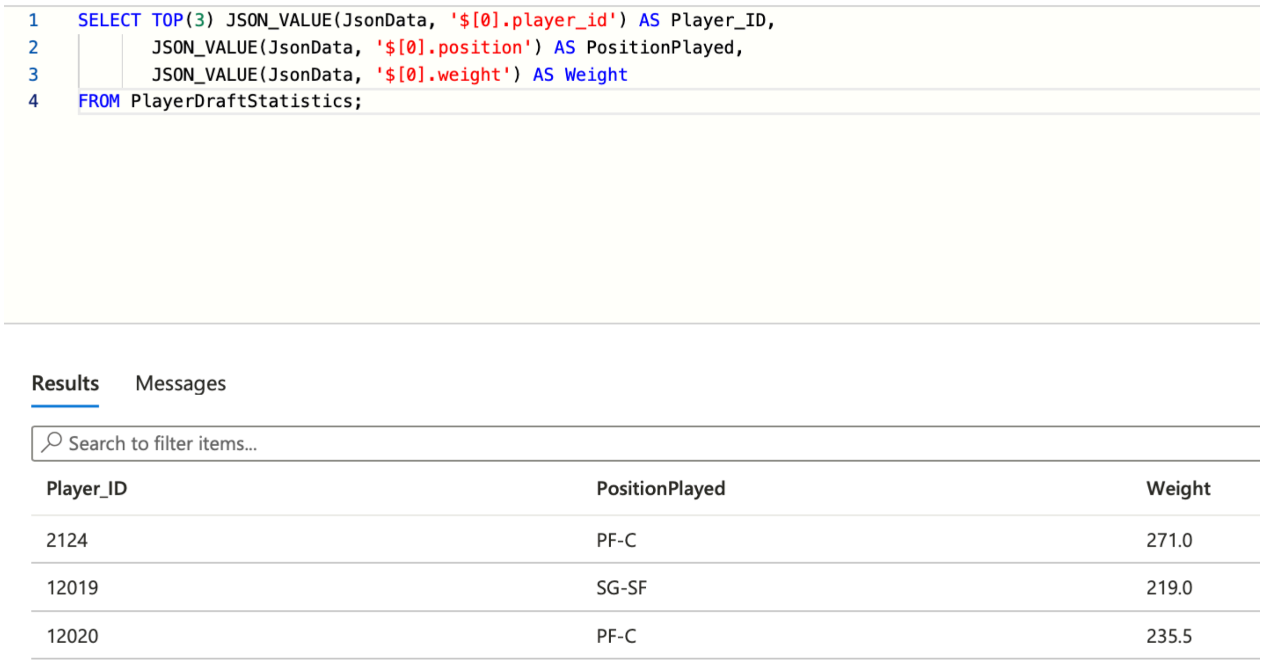
```
CREATE PROCEDURE usp_Insert_Plays_For
AS
BEGIN
    DROP TABLE plays_for
    CREATE TABLE plays_for AS EDGE
    INSERT INTO plays_for
    SELECT p.$node_id, t.$node_id
    FROM Player_dimension p
    INNER JOIN Player_Team_Bridge pt ON p.Player_id = pt.Player_id
    INNER JOIN Team t ON t.TeamID = pt.Team_id
END



CREATE PROCEDURE usp_Insert_Plays_In
AS
BEGIN
```

```
    DROP TABLE plays_in

    CREATE TABLE plays_in (H_A varchar(10)) AS EDGE

    INSERT INTO plays_in
    SELECT n1, n2, h_a
    FROM (
        SELECT t.$node_id n1, g_h.$node_id n2, 'Home' h_a
        FROM GameDimention g_h
        JOIN team t ON g_h.home_team_id = t.TeamID
        UNION
        SELECT t.$node_id, g_h.$node_id, 'Away'
        FROM GameDimention g_h
        JOIN team t ON g_h.away_team_id = t.TeamID
    ) a;
END
```

Sample Queries:

We have executed the below sample queries on our Foundation data layer – specifically on the document and graph DB to ensure that data accuracy and integrity is in place.

1. Document DB query:



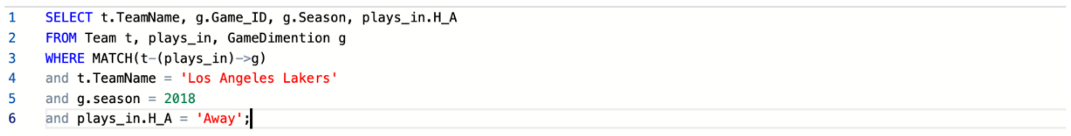
```
SELECT TOP(3) JSON_VALUE(JsonData, '$[0].player_id') AS Player_ID,
       JSON_VALUE(JsonData, '$[0].position') AS PositionPlayed,
       JSON_VALUE(JsonData, '$[0].weight') AS Weight
FROM PlayerDraftStatistics;
```

Results   Messages

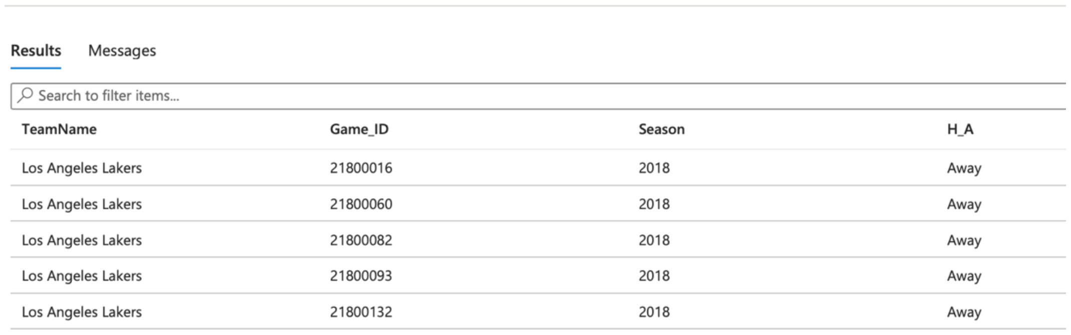
Search to filter items...

Player_ID	PositionPlayed	Weight
2124	PF-C	271.0
12019	SG-SF	219.0
12020	PF-C	235.5

2. Graph DB query:

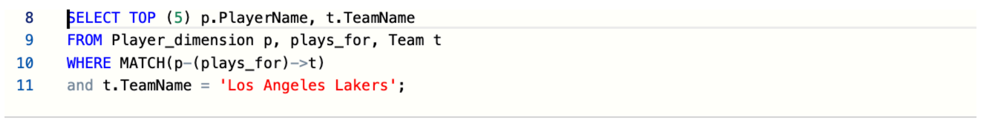


```
SELECT t.TeamName, g.Game_ID, g.Season, plays_in.H_A
FROM Team t, plays_in, GameDimention g
WHERE MATCH(t-(plays_in)->g)
and t.TeamName = 'Los Angeles Lakers'
and g.season = 2018
and plays_in.H_A = 'Away';
```

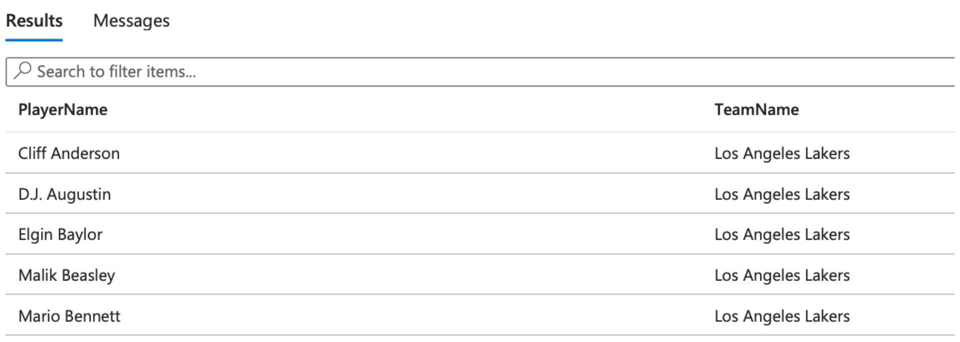


Results Messages

TeamName	Game_ID	Season	H_A
Los Angeles Lakers	21800016	2018	Away
Los Angeles Lakers	21800060	2018	Away
Los Angeles Lakers	21800082	2018	Away
Los Angeles Lakers	21800093	2018	Away
Los Angeles Lakers	21800132	2018	Away



```
SELECT TOP (5) p.PlayerName, t.TeamName
FROM Player_dimension p, plays_for, Team t
WHERE MATCH(p-(plays_for)->t)
and t.TeamName = 'Los Angeles Lakers';
```



Results Messages

PlayerName	TeamName
Cliff Anderson	Los Angeles Lakers
D.J. Augustin	Los Angeles Lakers
Elgin Baylor	Los Angeles Lakers
Malik Beasley	Los Angeles Lakers
Mario Bennett	Los Angeles Lakers

Next Steps:

As a part of next steps, specifically for P4 submission, we aim to automate the raw data store – i.e. figure out a way for the CSV to be uploaded to Azure Blob Storage automatically as and when the data changes in actual source – ie. Kaggle.

We also plan on creating a few reporting based views in the foundation layer for easy access for non-tech folks who do not want to wait for the reporting layer data – just as it would be in a real-world scenario.

****