Advanced SDQL Concepts:

You have made it through the SDQL basics page and have started to run your own queries! But likely you have just scratched the surface on that capabilities of SDQL and the kind of situations you want to explore.

This guide will cover several more advanced concepts that can help you run more complex queries and give you guidance on how to take things even further. Keep in mind that the possibilities for queries using SDQL are nearly endless!

Time Frames:

Any query that does not include a time frame will search the entire of that sports' database. However you can limit the time frame of your query using the *season* or *date* parameters. *day* (ie: day of the week) and *month* (January is 1, February is 2, etc.) also pertaining to isolating certain time frames.

Season is formatted by year and applies to the year in which the season starts (the 2020-21 NBA season is *season=2020*) Date is formatted as YYYYMMDD.

Example:

To query how NBA teams have faired since 2021-22 in March or later in the season after scoring at least 140 points last game:

p:points>=140 and season>=2021 and month>=3

To query all NFL games with a total over 51 between October 3, 2022 and January 1, 2023:

total>=51 and 20230101>=date>=20221003

A couple of notes:

- *date=today* is a special query that will query for games on the current date. You can also do simple functions on top of that. date=today-1 queries for yesterday, date=today+1 queries for tomorrow.
- You can isolate just the month and date if you want to look at performance on a certain date or multiple dates over more than one year.

  Example:

  To pull up all Christmas Day games: *int(str(date) [4 :]) =1225*

Open-Ended Queries (Undefined Parameter Values)

Thus far, all the queries have included parameters where the results are defined. However, a powerful research tool can be running queries where one or more of the parameters do not have a

defined result. The query output will then contain the game results for ALL possible values for that parameter.

Example:

To query how teams have performed the next game based on their previous margin:

p:margin

will provide a list of all possible results which you can then choose to sort by that value, record, number of games, etc.

You can also run a query with several parameters including an undefined value. To query how the Patriots have performed since 2020 when they won by more than 10 points last game based on the current game line:

team=Patriots and season>=2020 and p:margin>10 and line

You can also run a query with multiple undefined parameter values. To query how the Lakers performed in the 2022-23 season based on the final margin and number of assists they had in a game:

team=Lakers and season=2022 and margin and assists

EXPERT TIP: If you are not sure what the proper input or format for a given parameter. For instance, the simple query of *team* will provide a query that shows the proper name or abbreviation for each team.

Grouping

Similarly to open-ended queries, you can delimited values for a parameter using comma to see multiple results in a single query, grouped by value.

Examples:

To query how teams perform the game after covering by at least 5, 10, 15, and 20 points in their last game:

p:ats margin>=5,10,15,20

To query all combinations of how teams perform the game after covering by at least 5, 10, 15, and 20 points in their last game depending on if they scored at least 70, 80, 90 or 100 points last game:

p:ats margin>=5, 10, 15, 20 and p:points>=70,80,90,100

<u>Custom Table Queries</u> (Learning parameters @ conditions)

The standard query at Killersports.com comes with a preset list (now customizable!) of display columns for game and player queries. But the SDQL structure allows for you to customize queries to display any results you wish given a certain set of conditions using the parameters @ conditions.

To do this you can list every parameter you wish to display, each separated by commas. You then include the @ symbol, followed by the conditions of what you want displayed (think of the conditions section as what you would include in a standard query)

Examples:

To query the date and final margin of Mavericks games in the 2022-23 season:

date, margin @ team=Mavericks and season=2022

To query the date, points, passes, and completions in Colts home games in 2023:

date, points, passes, completions @ team=Colts and site=home and season=2023

Custom table queries also allow for open-ended queries and grouping

To see the points and rebounds for each NBA team in every game during the 2023 season:

points, rebounds @ team and season=2023

<u>Player Data</u>:

In addition to team-level data, you can search player-level data as well, either at the team level (see "Using Lists" for some examples on how you might do that) or by individual. Each sport has a set of parameters specific to players.

The standard format for querying player data is: Player Name:prefix:parameter

Examples:

To query for games following a game where LeBron James scored at least 40 points:

LeBron James:p:points>=40

To query for games where Patrick Mahomes threw for at least 300 yards and more than 2 touchdowns and the Chiefs won the last time he faced this team, use

Patrick Mahomes:P:passing yards>=300 and Patrick Mahomes:P:passing touchdowns>2 and P:W

<u>Game vs Player Queries</u>:

When you run those previous two or any other query with player data in a standard game query, the results show that player's team performance in the games that meet the criteria. However,

Killersports.com as has special player-query types to search for individual player performance. To switch query type, select your query type from the drop-down menu just about the query text bar.

Query Types by Sport:

NBA: Game, Player

MLB: Game, Batting, Pitching

NHL: Game, Player, Goalie

NFL: Game, Passing, Rushing Receiving, Defense, Fumbles, Interceptions, Kicking, Punting, Kick Returns, Punt Returns

To piggyback on the examples above, the query *LeBron James:p:points>=40* also works as a player query. However, this query will show the results for all players to play in games following a game where LeBron James scored at least 40 points. Naturally, you might want to see just LeBron's performance in such games. To do this use *name=LeBron James and LeBron James:p:points>=40* or just *name=LeBron James and p:points>=40*. Alternative you could query *name=Anthony Davis and LeBron James:p:points>=40* to see Anthony Davis' performance in games based on that LeBron statistic.

A couple of important notes:

- Name only works in player queries and will not work as a parameter in game queries
- In player queries, when there is overlap in a parameter between team and player data, the query will default to recognizing this as a player parameter. For example, the game query *LeBron James:p:points>=40 and p:points>=100* will provide games where LeBron scored at least 40 points and his team scored at least 100. That same query under the player tab will look for games where LeBron scored at least 40 points and someone on his team, LeBron or otherwise, scored at least 100 points (Spoiler: Such a game does not exist!)

Baseball-Only Prefixes

Because of the unique nature of starting pitchers in baseball (the importance of one player and the fact that the player is defined and listed well in advance of game time), SDQL provides two additional prefixs for the MLB in addition to the standard list of prefixs included in the SDQL basics:

s for the starter's last start
S for the starter's last start against the current opponent

Keep in mind that just because you are using a s prefix does not mean you are querying for a stat specific to the starter. *s:runs* refers to the amount of runs a starter's team had in the last game he started. *s:starter runs allowed* refers to the amount of runs a started allowed last start. MLB parameters includes several other starter specific-stat categories.

Using Lists

Database lists have the properties of Python lists. The contents of lists are accessed by offset (starting at 0) using square brackets.

As an example, inning runs is a parameter that is recorded in list form and inning runs[0] refers to runs in the first inning. inning runs[0]>=6 would query games where the team scored at least 6 runs in the first inning. You can also use build-in Python functions such as sum with lists. sum(inning runs[:5])>=10 refers to games where a team scored at least 10 runs in the first five innings (in this case, you can also use the helpful shortcut *S5*)

A helpful way to use lists in many cases is by sorting them, another Python function. sorted in lists all the list values from lowest to highest and you can then use the brackets to search in the opposite order (highest to lowest) in several instances. Player statistics in particular are recorded in lists.

Examples:

To query MLB teams coming off a game where they had at least 3 players with 3 or more hits:

sorted(list:p:hits) [-3]>=3

To query games where an NBA team did not have a scorer with 20 or more points:

sorted(list:p:points) [-1]<20

Average/Sum Aggregators

The following are SDQL aggregators

- **Average** can be abbreviated as **A**
- **Sum** can be abbreviated as **S**

Aggregators are most frequently used with the (parameters @ conditions) covered earlier in this page. The parameter *A(points)* or *Average(points)* queries based on the average number of points scored per game in the entirety of that sports database at the time of the current game. While there can be uses for that, it is much more likely that you will want to add some conditions to that search.

Average(points@season) is the query for the average amount of ppg scored for all teams in a given season.

Average(points@team and season) is the average number of ppg game scored by a team during the given season

Average(points@team and season and points>=130) is the average number of points scored by a team in a season in games where they scored at 130 points

Average(points@team and season and p:points>=130) is the average number of points scored by a team in a season in the games following games where they scored at least 130.

By far the most popular aggregators are Average and Sum There are also a couple of special use prefix to use with their abbreviations. The *t* prefix specifies both the team and season conditions. So *tA(points)* is a shorthand version of *Average(points@team and season)*. With that you can add p to the aggregator and in this case, p refers to the previous season instead of its standard use as previous game. tpA(points) in long form is *Average(points@team and season)[team and season-1]* Specific to baseball, s specifies *@starter and season*

Examples:

To query a team that has at least 3 40-point scoring games thus far this season:

tS(points>=40)>=3

To query a team with an average ATS margin of at least +1 ppg this season after having an average ATS margin of -1 ppg each of the past two seasons:

tA(ats margin)>=1 and tpA(ats margin)<=-1 and tppA(ats margin)<=-1

To query teams averaging more points per game on the road than at home this season:

tA(points@A)>tA(points@H)

To query teams averaging at least 3 points per game more than league average:

tA(points)-Average(points@season)>=3

Other Aggregators

- **Maximum** can be abbreviated as **Max**
- **Minimum** can be abbreviated as **Min**
- **Replace** can be abbreviated as **R**

Examples:

To query how the Celtics perform when they have scored at least 100 points in every game this season:

team=Celtics and Minimum(points@team and season)>=100

To query how a starter pitcher does in his first start after changing teams:

Replace(team@starter,N=1)!=team

N number of Games

You may have noticed the N=1 in that previous example that has not yet been covered. This is a condition that only looks back the amount of games specified. In the example above, we only want to look back to the starter's previous one start.

Examples:

To query teams that have an average line of -10 or better in their last 3 games:

tA(line,N=3)<=-10

To query when the Knicks have won at least 3 of their last 4 games by at least 10 points:

tS(margin>=10,N=4)>=3

Is statements

In most cases, = or *!=* work well for equal to or not equal to. However, queries can also be run with "is statements" *is* or *is not*. This is particularly important for querying something that has not happened yet in a given season or time frame.

Note: The null value when using aggregators is *None*. And because that is not a true value, is statements are needed for using None

Examples:

To query teams playing their first road game of the season (think of it as they have not played a road game yet)

A and tS(1@A) is None

To query team that have failed to cover by 20+ points in two straight games (where both of their last two games had lines):

tS(ats margin<=-20,N=2)=2 and p:line is not None and pp:line is not None

This is a type of query that might be important in the NCAABB database for instance, where not every game has a line

Conclusion

If you truly understand the concepts from the SDQL basics page and this page, you will likely be able to query 99+% of situations you could ever dream of. But even the experts at our database continue to learn new concepts. The best way to master SDQL is by continued experimenting and practice.

Good luck querying!