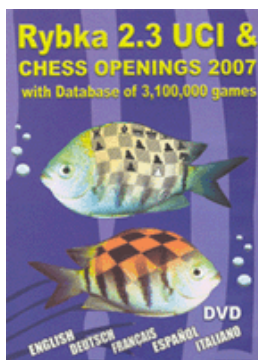




COLUMNISTS

*ChessOK
Cafe*

Dadi Jonsson



ChessCafe.com is pleased to invite readers to a game of chess at Convekta's ChessOK Playing Zone!



Click here for the [Flash](#) version or here to download and install the [Full](#) version.

Or play online against [Rybka](#).

CQL Queries in Chess Assistant

Chess Query Language (CQL) was designed by Gady Costeff and Lewis Stiller to specifically search for games, problems and studies that match certain themes. It is extremely powerful and CQL can find much more complex themes than traditional chess database systems.

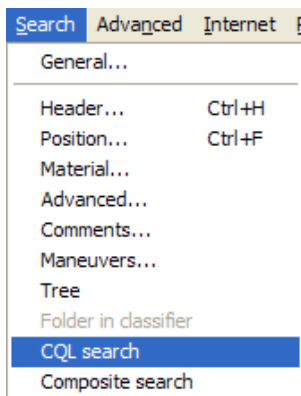
The current version of [Chess Assistant](#), as well as [Rybka & Chess Openings 2007](#), offers all the power of CQL in an integrated environment and brings several advantages, including:

- CQL queries can be run on any type of database that Chess Assistant can handle, instead of only PGN databases.
- CQL queries are not bound to a specific database. Define the query once and then apply it to any selection of games from any database.
- The result of a CQL query is displayed as a list of all the matching games with markers showing the exact position(s) that matched the query. Browsing through the resulting list of games will automatically display the first matching position in each game. Optionally, the matching positions can be written to an EPD file.
- When examining the matching positions on the screen, a diagram with automatically generated graphical annotations highlights the features that match the query.
- Chess Assistant allows you to organize your whole collection of queries into a neat structure, so you can store and quickly find any query you may be looking for. Each query is identified by a description and you can even attach an illustrative diagram, which facilitates the usage of this feature.

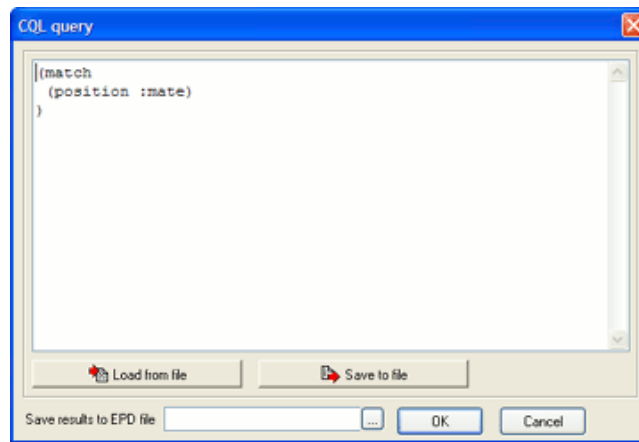
This article is an informal introduction to CQL, based on several simple examples that illustrate some of its capabilities. Although using and running CQL is quite simple in *Chess Assistant*, designing the queries themselves requires at least some basic knowledge of CQL. Fortunately, the manual is available [online](#) and contains many example queries that can serve as a starting point for experimenting.

Using CQL Search

In order to run a CQL query in Chess Assistant (or Rybka) open a database (Base > Open) and select a list of games (dataset) to search. By default a list showing all the games in the database is displayed. CQL will always search the games in the currently active list. Next, select "CQL search" from the Search menu.



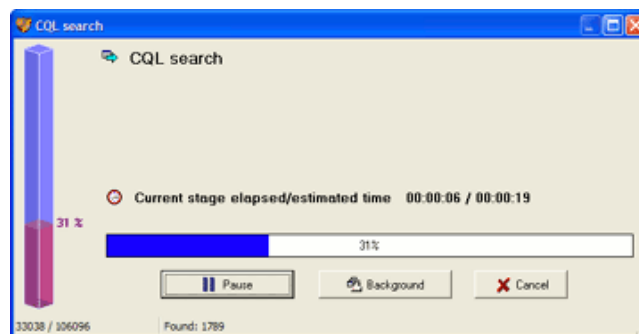
This displays the CQL query screen:



You can either type your query directly into the textbox (as was done here), or you can load a prepared query from a file using the “Load from file” button. The “Save to file” button allows you to save your query to a file at any time.

Some CQL queries can produce very interesting results and sometimes only the positions that match the query are of interest, but not the whole game. In such cases it is useful to save the matching positions to a file. In order to do so, either type the name of the file directly into the “Save results to EPD file” box or click on the button with the ellipsis to select a file.

When you are ready, click on OK and *Chess Assistant* will begin processing the query. In case you have made an error *Chess Assistant* displays an error message describing what’s wrong and allows you to continue editing the query.



While *Chess Assistant* is searching, a dialog box similar to this one is displayed. The colored bar on the left shows that 31% of the games have been searched, or 33,038 of 106,096 as shown below the bar. The text above the blue and white horizontal bar tells us that *Chess Assistant* has been searching for 6 seconds and the total search time is expected to be 19 seconds. The “Pause” button allows you to pause the search and continue later. The “Background” button minimizes the application and clicking on “Cancel” stops the search, but creates a list of the games found so far. This can be useful when testing new queries and allows you to quickly examine the results to see if they match your expectations. Finally the text “Found: 1789” at the bottom of the screen shows that so far 1,789 games (of the 33,038 already searched) match the query. Quite often these numbers will tell you if your query is actually doing what you expected. If you know that only a few games can match your query, but the statistics show that several hundred have been found, something must be wrong and it may be best to hit the “Cancel” button and check the results.

The result of a CQL query is returned as a list of games that matched the query. When you open one of the games you are taken directly to the first position where a match was found. In the image below, the diagram shows that the white bishop checkmated the black king. In the notation pane the matching move is indicated with a red “p.” Of course, there could be multiple markings when there is more than one matching position in a game.



The Structure of CQL Queries

A CQL query has the following generic structure:

```
(match
(position ... )
(position ... )
(position ... )
...
)
```

In this example, the “...” represent various keywords and commands. Each of the “(position ...)” elements is usually referred to as a position list.

Here is a simple CQL query to find games where White wins without moving his king (see explanations below):

```
(match
:result 1-0 ; White wins
(position :movefrom K :matchcount 0)
)
```

A keyword in CQL is a string beginning with “:.” There are three keywords in this example: “:result,” “:movefrom” and “:matchcount”:

:result – Note that this is the only keyword outside the position list as the result is a game attribute. This keyword has one parameter, 1-0, indicating that we are only interested in games won by White.

:movefrom – This keyword is within the position list, meaning that it contributes to the description of a particular position(s) that we are interested in. :movefrom takes a single parameter, a piece designator. “K” is a very simple piece designator. It means that we are looking for any move by the white king (“k” would be the black king). A slightly more complicated (and familiar) example of a piece designator would be “Kg4,” i.e. the piece designator can also specify a square (actually more than one) for the piece.

:matchcount – This keyword specifies a limit on how often the position we are describing can occur in the game. In our case there is one parameter specified “0.” It means that we are looking for games where the position described within the “(position ...)” never occurs. In other words, we are looking for games where the white king never moves.

Finally, the text “; White wins” is a comment. A comment starts with a semicolon and extends to the end of the current line.

Now, let’s have a look at a few more CQL examples.

Double Check

CQL can be helpful in chess training where you want to find games with some particular theme. Here is a simple example finding games where the white king is in double check:

```
(match
(position :attackcount a K 2)
)
```

Here we see a new keyword, :attackcount. It takes two piece designators (“a” and “K” in our example) followed by a range (here “2”) as parameters. The “a” stands for the attacking piece(s) and is shorthand for “any black piece” (including pawns), so it is equivalent to writing [kqrnp] (note the square brackets):

```
(position :attackcount [kqrnp] K 2)
```

The second parameter (“K”) stands for the white king. The range is “2” meaning that there must be two simultaneous attacks. Looking at the position list as a whole, we are looking for a position where two black pieces attack the white king. Instead of a single number, we could have specified a range for the number of attacks:

```
(position :attackcount a K 1 100)
```

Here we have replaced “2” with “1 100” meaning “from 1 to 100.” In other words this would match any position where the white king is in check.

Switching Colors

In the example above we looked for positions where there was a double attack on the white king. Similarly the following query would find positions where the black king is in double check:

```
(match
(position :attackcount A k 2)
)
```

The only difference is that now the attacking pieces are “A” (meaning any white piece) and the attacked piece is now “k,” i.e. the black king. But there is an easier and more general way to flip the colors automatically in a query:

```
(match
(position :attackcount a K 2 :flipcolor)
)
```

The `:flipcolor` keyword automatically tries the colors as they are specified and then checks for the same pattern with colors reversed. As a result this query would find all games where either king is in double check.

White Wins Without Castling

```
(match
:result 1-0
(position :movefrom Ke1 :moveto .[c1,g1] :matchcount 0)
)
```

We are already familiar with three of the keywords in this query: `:result`, `:movefrom` and `:matchcount`.

`:moveto` – This keyword takes a piece designator as a parameter, just like `:movefrom`. It describes the allowed destination square(s) of a move. In this example, the parameter of `:moveto` is `.[c1,g1]`. The `.”` stands for an empty square. So in this case we search for moves where the white king on e1 moves to an empty square. But not any empty square will do. It must be either the c1-square or the g1-square (`.[c1,g1]`). Of course the only way for a king to move from e1 to either of these squares is by castling.

So looking at `:movefrom` and `:moveto` we can see that the position list needs a castling move. But we also have `:matchcount 0` which means that a game will not match the query except if White never castles. Additionally, White must win the game (`:result 1-0`). As a result we find all games where White wins in spite of not castling. This may include games where White moves his king, as long as he never castles. For the case where White wins, but never moves his king see the first example in this article.

Search the Whole Board for a Pattern

One of the powerful features of CQL is that you can specify a pattern and then search for it anywhere on the board using the `:shift` keyword. Here is an example where we search for a white passed pawn.

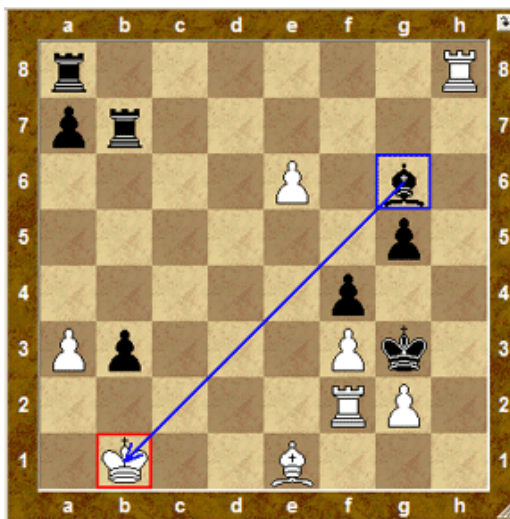
```
(match
(position
Pd2
:piececount [pP][d3-7] 0
:piececount p[c3-7,e3-7] 0
:shift
)
)
```

We start by putting a white pawn on d2 (“Pd2”). Next we see two lines with a new keyword, `:piececount`. It takes a piece designator and a range as a parameter. The piece designators we see here are more interesting than the ones we have seen so far, specifying both multiple

Meeting Check with a Checkmate

```
(match
 (position
  :sequence (
   (position :check)
   (position :mate)
  )
 )
)
```

Running this query in *Chess Assistant* on the three million game HugeBase returned 159 games. Here is a nice little example from the simul game Alekhine, A. - Popovic, A. (Osijek, 1930). Black just played **42...Bg6** giving a check, which Alekhine answered with **43 Rc2**, blocking the check and simultaneously checkmating the black king.



How often is the b-pawn promoted?

Even some of the simple queries we have looked at so far are impossible to do in traditional chess database systems. Here is one more such query, although slightly more complicated. We want to know how often the b-pawn is promoted. Of course, the b-pawn can theoretically be promoted on any file. It can even reach the h8-square in six moves. So when a pawn is promoted we need to figure out where it started from. A person walking past a chessboard and seeing a pawn being promoted would often have no idea which pawn it was unless he had followed the game from the beginning. If the pawns were marked in some way, it would be a different story and no matter where you saw a pawn on the board you would know which pawn it was. This is actually a technique offered by CQL to follow individual pieces around the board as the game progresses using the `:forany` keyword. Here is how:

```
(match
:forany pawn P
(position :initial $pawn[b2])
(position :movefrom $pawn :promote U)
)
```

The `:forany` keyword in a match list defines a tag and a piece designator. In our example, we are looking at the white pawns (“P”) and we have chosen the tag “pawn” as an identifier. So, “pawn” can stand for any white pawn on the board. If we want to limit it to a particular pawn, an additional condition is needed as we describe below. We can use the tag we have selected in the position lists in the query by preceding the name with a “\$” (“\$pawn”). Let’s see how that is done.

```
(position :initial $pawn[b2])
```

The `:initial` keyword ties this position list to the initial position of the game, and in that position our \$pawn must be on the b2-square. So, here we have ensured that \$pawn can only refer to the b-pawn.

When we have two or more position lists in a CQL query, it matches a game only if there is a match for all the position lists. Note that we did not use the `:sequence` keyword here, but we made sure that the first position list was tied to the initial position. And it will always match for a normal game of chess as there is always a b2-pawn in the initial position.

Now we come to the second position list:

```
(position :movefrom $pawn :promote U)
```

Since \$pawn refers to the pawn that started on b2, this position list matches only a move by that pawn which also is a promotion. The promotion part is ensured by the `:promote` keyword, which takes a single parameter, a piece designator. In this case it is “U,” which is shorthand for any piece at all. Therefore, it matches regardless what piece the pawn is promoted to.

I ran this query on HugeBase and it returned almost 14,000 games. In two games the pawn promoted on g8. Here is one of them.

Tseitlin, Mark (ISR) – Okunev, Taras (RUS)
Ramat Aviv (Israel), 2003

1.e4 e5 2.Nf3 Nf6 3.Nxe5 d6 4.Nc4 Nxe4 5.Nc3 Nxc3 6.bxc3 d5 7.Ne3 c5 8.Qf3 Be6 9.Rb1 Nc6 10.Rxb7 a6 11.e4 Nd4 12.Qg3 Rg8 13.cxd5 Bd6 14.dxe6 Bxg3 15.exf7+ Kf8



The pawn on f7 started on b2 and now it queens on g8 only sixteen moves into the game!

16.fxg8=Q+ Kxg8 17.hxg3 Qe8 18.Bd3 h6 19.O-O Qc6 20.Rb1 Kh8 21.Bb2 Ne6 22.Bc3 Rd8 23.Rfe1 Nd4 24.Nf5 1-0

By doing a similar search for other pawns you can find out which pawn is most often promoted!

Knight Gives Mate with its First Move

Games in which a knight gives mate with its first move are rare. Edward Winter has collected and published several such games, including four specimens in his book [Kings, Commoners and Knaves](#) (Russell Enterprises, Inc., Milford, 1999, pages 78-80).

```
(match
:forany knight N
(position :movefrom $knight :matchcount 1)
(position
:mate
:attackcount $knight k 1
)
)
```

Here we use tagging again, this time to keep track of the white knights. The first position list ensures that we only take into consideration a knight that moves exactly once during the game. The second position list ensures that the knight attacks the black king in the final position where the king is checkmated.

Running this query through HugeBase returns nine games. It is easy to modify the query so that it finds similar checkmates by Black.

Do You Have a Query?

If you have created some interesting CQL queries, I would be interested in seeing them and possibly publishing them in a future column. Also, if you have given up trying to search for an interesting theme using traditional chess database tools. Send me a note describing what you were trying to search for and I might come up with a solution in CQL.

Chess Assistant supports the full Chess Query Language and adds several extensions. Some of the queries that were taken as examples in this article are quite simple, but others use more advanced features like tagging. There are features in *Chess Assistant* that make managing CQL queries much easier than were not described here. In particular I would like to point the interested reader to the Composite Search function.

All the Chess Assistant software described by Dadi in this column, as well as many more Chess Assistant programs, are available in the [USCF Sales Online Catalog](#).

Dadi wants your questions!! Send it along and perhaps it will be answered in an upcoming column. Please include your name and country of residence. [Yes, I have a question for Dadi!](#)



[TOP OF PAGE](#)



[HOME](#)



[COLUMNS](#)



[LINKS](#)



[ARCHIVES](#)



[ABOUT THE
CHESS CAFE](#)

[\[ChessCafe Home Page\]](#) [\[Book Review\]](#) [\[Columnists\]](#)
[\[Endgame Study\]](#) [\[The Skittles Room\]](#) [\[Archives\]](#)
[\[Links\]](#) [\[Online Bookstore\]](#) [\[About ChessCafe.com\]](#) [\[Contact Us\]](#)

© 2007 CyberCafes, LLC. All Rights Reserved.

"**ChessCafe.com®**" is a registered trademark of Russell Enterprises, Inc.