# Topics in English Syntax: CQL and Sketch Engine

Doug Arnold
Language & Linguistics
doug@essex.ac.uk

## 1 Introduction

CQL stands for 'Corpus Query Language' and it provides the most general way of searching SketchEngine (and other tools for corpus search). It is quite simple.

## 2 Queries

The basic rules are very simple:

- every element of a query should be in square brakets;
- the search term should be in quotations "bank"
- you should specify the type of search, e.g. do want to search for "bank" as a lemma or a word?
- you can combine search terms in a single element, e.g. specify that you are interested in the word "bank" when it is tagged as a VVG (an *-ing* form); the search terms are combined with &;
- you can look for consecutive having searches with several elements
- as well as equations, you can have 'inequations' (not equal)
- you can use 'wildcards' (regular expressions) in most places

Here are some simple examples:

(1) a. `[word="bank"]`
b. `[lemma="bank"]`
c. `[word="bank" & tag="VV"]`
d. `[word="bank" & tag="VV"] [word = "on"]`
e. `[word="bank" & tag="VV"] [word != "on"]`
f. `[word="bank" & tag="VV"] [] [word="on"]`
g. `[word="bank" & tag="VV"] [][] [word="on"]`

SketchEngine allows some simplications for some of this:

Normally, in CQL, every item in the search should be in square brackets, but SketchEngine seems not to insist on this, so you can write (2a) in place of (2b)

(2) a. `"Ought"`
b. `["Ought"]`

Normally in CQL you must specify what you are searching for (e.g. word, lemma, part of speech tag), but again, SketchEngine does not insist. If you select 'CQL query', and look to the right, you will see a 'Default attribute' box. The value selected here determines what a query like the following will mean:

(3) `["be"]` or `"be"`
(4) `[lemma="be"]`
(5) `[word="be"]`

E.g. if you have "lemma" selected, you will get back forms of "be". For example, if you select word, and search for "is", you will get many results, but if you select "lemma" and do the same search, you will get nothing (there is no lemma called 'is', the lemma is 'be').

If you select "tag", your search will be interpreted a search for items that are tagged in the way indicated. For example, the tag "IN" is used in this version of the BNC for prepositions. So the following (equivalent) queries will return examples involving all kinds of prepositions:

- `[tag="IN"]`

- `["IN"]` (with default attribute tag selected)
- `"IN"` (with default attribute tag selected)

But selecting word as the default attribute gives examples involving the word *IN* (upper case). If you search for this as a lemma, you get the empty result (the lemma is called 'in' – i.e. in lower case).

The other attributes you can select as default are "lemmpos" – this is a combination of the lemma and a shortened part of speech tag, e.g. "bank-v" corresponds to the lemma "bank" and the part of speech tags for verbs.

If in doubt, you can always specify these things explicitly (I usually do).

You can use regular expressions (see below) in the names of tags and lemmas. For example, I can never remember what the tags for different kinds of verb are, but I remember that they almost all begin with "V" (e.g. "VBD", "VBG", etc.) so if I want to find instance of the word *bank* that have been tagged as a word, I will typically write:

(6) `[tag="V.*" & lemma="bank"]`

Since I am not only too stupid to remember the tags, I am also too lazy to look them up, I often just do a search to see what tag I want. If you look under the menu item 'Frequency' you can select 'Node tags', this will show you the tags that have been used (and their frequency).

(You can do the same thing by selecting 'View Options' (*not* 'KWIK' or 'Sentence'), and playing with the menu you see).

You can also use regular expressions on query elements:

(7) `[word="the"] []* [word="bank"]`
(8) `[word="very"]{3,20}`

## 3 Regular Expressions

The real power (and fun) of using CQL is that it allows you to carry out searches using 'regular expressions'.

A Regular Expression is a string of characters (an expression) which is interpreted as a pattern which describes or matches other strings. They are a feature of many programming languages. The general idea is always the same, but the details often vary slightly. Queries using them provide a very flexible and powerful way of searching.

Regular Expressions have a *syntax* and a *semantics*.

### 3.1 Syntax

A pattern (Regular Expression) is a string of characters which is used as a template or pattern. Such a string can contain three kinds of character:

**literal characters** , such as "`a`", "`b`" or "`c`". These match occurrences of the same character.
**meta-characters** are special character or character sequences which match some other characters. In particular, the dot "`.`" is a metacharacter which matches any single character.
**special characters** are characters which affect how the meta or literal characters are to be interpreted: these are the square brackets, "`[`" and "`]`", the parentheses "`(`" and "`)`", braces `{` and `}`, the caret "`^`", the repetition operators "?", "*" and "+", the disjunction symbol "`|`", and the backslash "\".

### 3.2 Matching ('Semantics')

The 'meaning' of a regular expression (pattern) is what it matches. Here are the rules for matching:

- a literal character matches itself: "`d`" matches *d*; as we will see, patterns can be concatenated, so "`do`" and "`dog`" are patterns, matching respectively *do* and *dog*.
- a dot "`.`" matches any character. Thus, `fe.t` matches *feet* and *feat;*
- "`?`", "`+`", and "`*`", and `{n,m}` (where `n` and `m` are numbers) can follow a pattern to indicate repetition. Suppose "`X`" is a pattern, then:
  - "`X?`" matches when "`X`" matches one or zero times ("`X`" is 'optional'), e.g. "`colou?r`" matches *color* and *colour;*
  - "`X*`" matches zero or more times;
  - "`X+`" matches one or more times.
  - "`X{3,5}`" matches between 3 and 5 five times.

  **Examples**:

| pattern | matches | does *not* match |
|---|---|---|
| `ft` | ft | fet, feet, feeet, . . . |
| `fet` | fet | ft, feet, feeet, . . . |
| `f.t` | fet, fat, fit, fft, . . . | ft, feet, feat, . . . |
| `f..t` | feet, feat, faet, fxzt, . . . | ft, feeet, . . . |
| `fe*t` | ft fet feet, feet, feeet, . . . | |
| `fe+t` | fet feet, feet, feeet, . . . | ft |
| `fe?t` | ft, fet | feet, feet, feeet, . . . |
| `fe{0,3}t` | ft, fet, fee, feeet | feeeet, feeeeet, . . . |

- A sequence of characters within square brackets matches any one of them, e.g. "`[aeiou]`" matches any vowel;
- Within square brackets, a hyphen can be used to express a range. For example, the patterns "`[0-9]`" and "`[0123456789]`" are equivalent: either one will match any digit.
- Within square brackets at the start of a sequence, the caret "`ˆ`" is a special character indicating that any character *not* in the sequence should be matched. For example, the pattern "`[ˆaeiou]`" will match any consonant (non-vowel); the pattern "`[ˆ0-9]`" will match anything which is *not* a digit.
- Otherwise, within square brackets, characters are interpreted literally. Thus, "`[.?]`" will match a dot followed by a question mark.
- Two or more patterns can be concatenated to match what the first matches followed by what the second matches. E.g. "`[aeiou][xyz]`" matches any vowel followed by any of "`x`", "`y`", or "`z`".
- Two or more patterns can be combined as alternatives using the disjunction meta-character (a vertical bar), e.g. "`seek|sought`" will match either the word *seek* or the word *sought* .
- Any character preceded by the backslash ("\") is treated as a literal (even if it is a meta-character); e.g. "`3\.1`" will match *3.1*, and "`who\?`" will match *who?*. A backslash has no effect with normal characters (so "\a" is the same as "a"). To match an actual backslash, the pattern "\\" can be used.
  Compare:

| pattern | matches | does *not* match |
|---|---|---|
| `f\.t` | f.t | fit, . . . |
| `f.t` | f.t, fit, . . . | |

- Parentheses "`()`" can be used to group parts of a pattern together: for example, matches for *seek* and *saught* can be found with the pattern "`s(eek|ought)`".

There is one final aspect of regular expressions that is worth mentioning (though it is not particularly useful in practice). It is possible to refer back to earlier patterns within a term by writing `\1` or `\2`, etc. This refers back to the nth previous bracketed expression. (91) and (9b) mean the same: any sequence of four letters – any four letter word. (9c) means a word consisting of four letters followed by *the same* four letters, (e.g. *couscous*). Notice you can only use this within words (not across sequences of word).

(9) `[word="[a-z]{4}"] [word="([a-z]{4})"] [word="([a-z]{4})\1"]`

## 3.3 Exercises: Warm Up

Hint about writing regular expression: You will make mistakes with patterns – you often only get the right pattern after a number of attempts, so use Notepad or similar program to compose the patterns, and cut and paste.

Hint: Don't try to get the right pattern straight off – trial and error is a good method for this kind of thing.

**Exercise 3.1** In the following, do the patterns in the first column match the words in the second?

| pattern | word | matches? |
|---|---|---|
| `f.?t` | ft | |
| `f.?t` | fit | |
| `f.?t` | feet | |
| `f.*t` | ft | |
| `f.+t` | ft | |
| `f.*t` | feeeeet | |
| `f.+t` | feeeeet | |

**Exercise 3.2** Write a pattern that will find single letter words.

**Exercise 3.3** For reasons best known to yourself, you are interested in four letter words. Write a pattern that will match such words.

**Exercise 3.4** Refine this a bit: write a pattern that will match four letter words that rhyme with "*luck*".

**Exercise 3.5** Write a query that will find five letter words that rhyme with "*luck*".

**Exercise 3.6** Write query that will find for four and five letter words that rhyme with "*luck*".

# 4 Exercises: Assessment

These exercises are the second (of three) pieces of assessed work. You should just hand in the answers (soft or hard copy), the answers will typically be very brief, e.g. the answer to the second exercise above would be `[word="."]`. Where there is an 'extension' exercise, it is not part of the assignment.

**Exercise 4.1** Write a pattern that will match both British and American spellings of "*colour*" and "*colours*", i.e. "*colour*", "*colours*", "*color*", "*colors*".

**Exercise 4.2** Write a pattern that will find words that end in "*ought*". (Extension exercise: when you think of "*thought*", "*fought*", "*ought*", etc., you may think they are all pronounced to rhyme with "*short*". But is this really true? [hint: look at examples starting with "*d*"]).

Write a pattern that will find words that begin with "*d*" and end in "*ought*"

**Exercise 4.3** The following are pairs of words that prescriptive grammars claim are often confused. Write patterns that will match both members of the pair (imagine you are collecting a subcorpus of examples for later study):

- "*beside*" or "*besides*".
- "*dependent*" or "*dependant*"..
- "*ingenuous*" or "*ingenious*"..
- "*affect*" or "*effect*".
- "*to*" or "*too*".

- *"to"* or *"two"*.

**Exercise 4.4** The following are some more words that are often confused (according to prescriptive grammars, cf. exercise 4.3). Write patterns that will match both members of the pair:

- *"contemptible"* or *"contemptuous"*.
- *"continual"* or *"continuous"*.
- *"council"* or *"counsel"*.
- *"deprecate"* or *"depreciate"*.
- *"imperial"* or *"imperious"*.
- *"judicial"* or *"judicious"*.
- *"luxuriant"* or *"luxurious"*.
- *"precipitate"* or *"precipitous"*.
- *"reverand"* or *"reverent"*.

[Extension exercise: check the 'official' definitions in a dictionary. Try to find pairs of examples that show the differences for one or two of the pairs].

**Exercise 4.5** Explain the difference between the patterns "`[az]`" and "`[a-z]`". Do a search to check.

Do the same for "`[ac]`" and "`[a-c]`".

**Exercise 4.6** Write patterns that will match the following:

- single digit numbers (e.g. *"9"*, *"2"*, etc.)
- double digit numbers (*"93"*, *"29"*, etc.)
- triple digit numbers (*"297"*, *"296"*, etc.)
- numbers consisting of one or more digits (*"9"*, *"29"*, *"236"*, etc).
- numbers consisting of two or more digits (*"29"*, *"236"*, etc).
- numbers consisting of three or more digits (*"297"*, *"2368"*, etc).
- numbers consisting of digits and commas (*"7,300"*, *"26,368"*, etc).
- numbers containing a decimal point (*"9.8"*, *"29.2"*, *"236.0"*, etc.)
- numbers containing a decimal point and commas (*"9,521.8"*, *"2,976.2"*, etc.)

**Exercise 4.7** What is the longest sequence of *very*s in the BNC? Write a pattern that will find it.

**Exercise 4.8** How many difference spellings can you find for *"manoeuvre"* in the BNC?

Which is most common? Hint: start with a pattern that is quite flexible, and narrow it down.

**Exercise 4.9** Are there any words in the BNC that consist only of vowels (i.e. only a, e, i, o, and u)? Write a pattern to check.

**Exercise 4.10** Are there any words in the BNC that contain *no* vowels (i.e. none of a, e, i, o, and u)? Write a pattern to check.

**Exercise 4.11** The consonants of English are `[bcdfghjklmnpqrstvwxz]` (i.e. the letters, minus `[yaeiou]`). Write patterns that will find words with:

- a cluster of four consonants word initially;
- a cluster of four consonants word finally;
- a cluster of four consonants word medially.

**Exercise 4.12** Is there any difference between the patterns "`qu`" and "`q[u]`"?

**Exercise 4.13** Is there any difference between the patterns "`qu*`" and "`q[u]*`"?

**Exercise 4.14** Explain the difference between the patterns "`q[u]*`" "`q[u].*`"

**Exercise 4.15** One of the earliest spelling rules I learned was that *q* is always followed by *u*. Are there any words in the BNC that consist of a *"q"* which is *not* followed by a *"u"*? Devise a query to find out.

Most of the results you get will be abbreviations, or other oddities. Try to find ways of excluding them.

**Exercise 4.16** The following are some words that have alternative spellings. Write patterns that will match either spelling (you could use this to check, e.g. relative frequencies and other patterns of grammatical usage).

- *"paralyse"* or *"paralyze"*.
- *"kilogram"* or *"kilogramme"*.
- *"cheque"* or *"check"*.
- *"program"* or *"programme"*.
- *"there"* or *"their"*.
- *"amid"* or *"amidst"*.
- *"draught"* or *"draft"*.

**Exercise 4.17** The following are among the English verbs have alternative past participle forms. Write patterns that will match either:

- *"kneeled"* or *"knelt"*.
- *"leaped"* or *"lept"*.
- *"spilled"* or *"spilt"*.
- *"spelled"* or *"spelt"*.
- *"learned"* or *"learnt"*.
- *"dreamed"* or *"dreamt"*.
- *"burned"* or *"burnt"*.
- *"spoiled"* or *"spoilt"*.

**Exercise 4.18** Returning to the words that contain no vowels (exercise 4.10):

- Write a pattern that will find words that contain no vowels.
- Some of the results you get will be numbers. Write a pattern that would also exclude them.
- Some of the remaining results are words consisting of a single letter. Refine the pattern to exclude them.
- Some of the results consist of a letter and a 'dot', or apostrophe. Refine the pattern to exclude them.
- The 'words' = and > are probably in the results. Refine the pattern to get rid of them.

**Exercise 4.19** Write patterns that will match:

- *"is"* or *"are"*.
- *"was"* or *"were"*.
- *"is"* or *"are"* or *"was"* or *"were"*
- any uncontracted form of the verb *"to be"*, i.e. *"am" "is"*, *"are"*, *"was"*, *"were"*, *"be"*, *"being"*, *"been"*;
- any uncontracted form of the verb *"to be"*, followed by *"happy"*;
- the contracted form of *"am"*, i.e. *"'m"*;

**Exercise 4.20** You learn in phonology that some affixes, such as *"-ic"*, affect stress, cf.

- *"as'tronomy" "astro'nomic"*
- *"pho'tography" "photo'graphic"*

Suppose your phonology assignment involves investigating other affixes to see if they affect stress (always, or sometimes). Write a pattern that will allow you to gather a sample of words that end in the following suffixes as a basis for further study:

- *"-ness"*
- *"-ess"*

- *"-ise"/"-ize"*

**Exercise 4.21** Many English prefixes occur with and without hyphens, e.g. *"crypto-fascist"*, *"cryptographic"*. Devise a pattern that will find forms of *crypto* with or without the hyphen. Do the same for *"mal-"*.

**Exercise 4.22** There are several English prefixes that mean roughly *"four"*: *"four-"*, *"quad-"/"quat-"*, and *"tetra-"*. Write a pattern that could be used to compile a corpus of examples which will allow you to explore whether there are any systematic differences.

The following are some other prefixes with similar meanings. Do the same for them.

- *"many-"*, *"multi-"*, and *"poly-"*;
- *"super-"*, *"hyper-"*, and *"ultra-"*.
- *"uni-"*, and *"mono-"*.
- *"hemi-"*, and *"semi-"*.

**Exercise 4.23** Can prefixes like *"pre-"* and *"ex-"* occur as independent words? With and without hyphens? Write patterns that will allow this to be investigated.

**Exercise 4.24** Write a query that will find instances of prefixes that have been conjoined as independent words, like *pre- and post- war*.

**Exercise 4.25** Does the BNC contain any words that contain a backslash? ('Words' like `\windows`?). Write a pattern that will allow you to find this?

## 5 A Few Extra Notes

A couple of things to bear in mind.

- Corpus tagging is generally done automatically (with some hand checking afterwards). This gives amazingly good results – but not perfect results. Beware.
- In some cases, the notion of a word that has been used for tokenizing a corpus (i.e. splitting it into words) may be different from yours or mine. For example, in the BNC looking for forms like *"aren't"* with a query like the following will give disappointing results:[1]
  - `[word="no"] [word="matter"] [word="how"]`
  - `[word="aren't"]`

  This is because *"aren't"* is tokenised as two words: "are" an "n't", so the query should be:
  - `[word="are" [word="n't"]`
- Finally, note that there are some limitations to what can be expressed in regular expressions. It's easy enough to search for words that consist of a string of characters followed by the same string of characters:

  (10) `[word="(.+)\1"]` (one or more characters followed by the same sequence)

  But it is *impossible* to find words consisting of a sequence followed by the same sequence in reverse (palindromes, e.g. *noon*).

  Of course, one can find four or six letter words like this:

  (11) a. `[word = "(.)(.)\2\1"]` (n o o n)

  b. `[word = "(.)(.)(.)\3\2\1"]` (r e d d e r)

  But there is no general method.

[1]On the other hand, there used to be cases where forms like *"in order"*, *"in part"*, and *"no matter how"* — that is, expressions that contain spaces — were treated as single items (e.g. as `no_matter_how` ). So the following search would not find you examples of "no matter how". **But this seems to have been fixed now**.