

Contents

1 Guide for qjanno v1.0.0

1.1	Background	1
1.2	How does this work?	1
1.3	The CLI interface	1
1.3.1	CLI details	3
1.3.2	The <code>-c/--showColumns</code> option	4
1.4	Query examples	4

1 Guide for qjanno v1.0.0

1.1 Background

qjanno is a fork of the [qhs](#) software tool, which is, in turn, inspired by the CLI tool [q](#). All of them enable SQL queries on delimiter-separated text files (e.g. `.csv` or `.tsv`). For qjanno we copied the source code of qhs v0.3.3 (MIT-License) and adjusted it to provide a smooth experience with a special kind of `.tsv` file: The Poseidon [.janno](#) file.

Unlike `trident` or `xerxes` qjanno does not have a complete understanding of the `.janno`-file structure, and (mostly) treats it like a normal `.tsv` file. It does not validate the files upon reading and takes them at face value. Still `.janno` files are given special consideration: With the `d(...)` pseudo-function they can be searched recursively and loaded together into one table.

qjanno still supports most features of qhs, so it can still read `.csv` and `.tsv` files independently or in conjunction with `.janno` files (e.g. for `JOIN` operations).

1.2 How does this work?

On startup, qjanno creates an [SQLite](#) database [in memory](#). It then reads the requested, structured text files, attributes each column a type (either character or numeric) and writes the contents of the files to tables in the in-memory database. It finally sends the user-provided SQL query to the database, waits for the result, parses it and returns it on the command line.

The query gets pre-parsed to extract file names and then forwarded to an SQLite database server via the Haskell library [sqlite-simple](#). That means qjanno can parse and understand basic SQLite3 syntax, though not everything. [PRAGMA functions](#), for example, are not available. The examples below show some of the available syntax, but they are not exhaustive. Trial and error is recommended to see what does and what does not work. Please report missing functionality in our [issue board on GitHub](#).

1.3 The CLI interface

This is the CLI interface of qjanno:

```
Usage: qjanno [--version] [QUERY] [-q|--queryFile FILE] [-c|--showColumns]
          [-t|--tabSep] [--sep DELIM] [--noHeader] [--raw] [--noOutHeader]
Command line tool to allow SQL queries on .janno (and arbitrary .csv and .tsv)
files.
```

```

38 Available options:
39   -h,--help           Show this help text
40   --version           Show qjanno version
41   QUERY               SQLite syntax query with paths to files for table
42                       names. See the online documentation for examples. The
43                       special table name syntax 'd(path1,path2,...)' treats
44                       the paths (path1, path2, ...) as base directories
45                       where .janno files are searched recursively. All
46                       detected .janno files are merged into one table and
47                       can thus be subjected to arbitrary queries.
48   -q,--queryFile FILE Read query from the provided file.
49   -c,--showColumns    Don't run the query, but show all available columns
50                       in the input files.
51   -t,--tabSep         Short for --sep $'\t'.
52   --sep DELIM         Input file field delimiter. Will be automatically
53                       detected if it's not specified.
54   --noHeader          Does the input file have no column names? They will
55                       be filled automatically with placeholders of the form
56                       c1,c2,c3,...
57   --raw               Return the output table as tsv.
58   --noOutHeader       Remove the header line from the output.

59 This help can be accessed with qjanno -h. Running qjanno without any parameters does not work: The QUERY
60 parameter is mandatory and the tool will fail with Query cannot be empty.

61 A basic, working query could look like this:

62 $ qjanno "SELECT Poseidon_ID,Country FROM d(2010_RasmussenNature,2012_MeyerScience)"
63 .------.------.
64 |   Poseidon_ID   | Country |
65 :=====:=====:
66 | Inuk.SG         | Greenland |
67 | A_Mbuti-5.DG    | Congo    |
68 | A_Yoruba-4.DG   | Nigeria  |
69 | A_Sardinian-4.DG | Italy    |
70 | A_French-4.DG   | France   |
71 | A_Dinka-4.DG    | Sudan    |
72 | A_Ju_hoan_North-5.DG | Namibia |
73 '-----'-----'

74 qjanno is asked to run the query SELECT ... FROM ..., which triggers the following process:

75 1. As d(...) is provided in the table name field (FROM), qjanno searches recursively for .janno files in the
76    provided base directories 2010_RasmussenNature and 2012_MeyerScience.
77 2. It finds the .janno files, reads them and merges them (simple row-bind).
78 3. It writes the resulting table to the SQLite database in memory.
79 4. Now the actual query gets executed. In this case the SELECT statement includes two variables (column
80    names): Poseidon_ID and Country. The database server returns these two columns for the merged .janno

```

81 table.
 82 5. qjanno returns the resulting table in a neat, human readable format.

83 1.3.1 CLI details

84 qjanno can not just read .janno files, but arbitrary .csv and .tsv files. This option is triggered by providing file
 85 names (relative paths) in the FROM field of the query, not d(...).

```
86 $ echo -e "Col1,Col2\nVal1,Val2\nVal3,Val4\n" > test.csv
87 $ qjanno "SELECT Col2 FROM test.csv"
88 .-----.
89 | Col2 |
90 :=====:
91 | Val2 |
92 | Val4 |
93 '-----'
```

94 qjanno automatically tries to detect the relevant separator of files. With --sep a delimiter can be specified
 95 explicitly, and the shortcut -t sets --sep \$('t' for tab-separated files. So a .janno file can also be read without
 96 d(...) using the following syntax:

```
97 $ qjanno "SELECT Poseidon_ID,Country FROM 2010_RasmussenNature/2010_RasmussenNature.janno" \
98   -t # -t is optional
99 .-----.-----
100 | Poseidon_ID | Country |
101 :=====:=====:
102 | Inuk.SG     | Greenland |
103 '-----'-----'
```

104 The --noHeader option allows to read files without headers, so column names. The columns are then automatically
 105 named c1,c2,...cN:

```
106 $ echo -e "Val1,Val2\nVal3,Val4\n" > test.csv
107 $ qjanno "SELECT c1,c2 FROM test.csv" --noHeader
108 .-----.-----
109 | c1 | c2 |
110 :=====:=====:
111 | Val1 | Val2 |
112 | Val3 | Val4 |
113 '-----'-----'
```

114 The remaining options concern the output: --raw returns the output table not in the neat, human-readable
 115 ASCII table layout, but in a simple .tsv format. --noOutHeader omits the header line in the output.

```
116 $ echo -e "Col1,Col2\nVal1,Val2\nVal3,Val4\n" > test.csv
117 $ qjanno "SELECT * FROM test.csv" --raw --noOutHeader
118 Val1 Val2
119 Val3 Val4
```

120 Note that these output options allow to directly prepare individual lists in trident's forgeScript selection language
 121 format:

```

122 $ qjanno "SELECT '<||Poseidon_ID||>' FROM d(2012_MeyerScience)" --raw --noOutHeader
123 <A_Mbuti-5.DG>
124 <A_Yoruba-4.DG>
125 <A_Sardinian-4.DG>
126 <A_French-4.DG>
127 <A_Dinka-4.DG>
128 <A_Ju_hoan_North-5.DG>

```

1.3.2 The -c/--showColumns option

-c/--showColumns is a special option that, when activated, makes qjanno return not the result of a given query, but an overview table with the columns available in all loaded tables/files for said query. That is helpful to get an overview what could actually be queried.

```

133 $ echo -e "Col1,Col2\nVal1,Val2\nVal3,Val4\n" > test.csv
134 $ qjanno "SELECT * FROM test.csv" -c
135 .----- .----- .-----
136 | Column | Path | qjanno Table name |
137 :=====:=====:=====
138 | Col1 | test.csv | test |
139 | Col2 | test.csv | test |
140 '-----'-----'-----

```

This summary also includes the artificial, structurally cleaned table names assigned by qjanno before writing to the SQLite database. Often we can not simply use the file names as table names, because SQLite has strict naming requirements. File names or relative paths are generally invalid as table names and need to be replaced with a tidy string. These artificially generated names are mostly irrelevant from a user perspective – except a query involves multiple files, e.g. in a JOIN operation. See below for an example.

1.4 Query examples

The following examples show some of the functionality of the SQLite query language available through qjanno. See the [SQLite syntax documentation](#) for more details.

Sub-setting with WHERE

Get all individuals (rows) in two Poseidon packages where UDG is set to ‘minus’.

```

151 $ qjanno " \
152 SELECT Poseidon_ID,UDG \
153 FROM d(2010_RasmussenNature,2012_MeyerScience) \
154 WHERE UDG = 'minus' \
155 "
156 .----- .-----
157 | Poseidon_ID | UDG |
158 :=====:=====
159 | Inuk.SG | minus |
160 '-----'-----

```

Get all individuals where Genetic_Sex is not ‘F’ **and** Country is ‘Sudan’.

```

162 $ qjanno " \
163 SELECT Poseidon_ID,Country \
164 FROM d(2010_RasmussenNature,2012_MeyerScience) \
165 WHERE Genetic_Sex <> 'F' AND Country = 'Sudan' \
166 "
167 .----- .-----
168 | Poseidon_ID | Country |
169 :=====:=====:
170 | A_Dinka-4.DG | Sudan |
171 '-----'-----'

```

172 Get all individuals where the the UDG column is not NULL **or** the Country is 'Sudan'.

```

173 $ qjanno " \
174 SELECT Poseidon_ID,Country \
175 FROM d(2010_RasmussenNature,2012_MeyerScience) \
176 WHERE UDG IS NOT NULL OR Country = 'Sudan' \
177 "
178 .----- .-----
179 | Poseidon_ID | Country |
180 :=====:=====:
181 | Inuk.SG | Greenland |
182 | A_Dinka-4.DG | Sudan |
183 '-----'-----'

```

184 Get all individuals where Nr_SNPs is equal to or bigger than 600,000.

```

185 $ qjanno " \
186 SELECT Poseidon_ID,Nr_SNPs \
187 FROM d(2010_RasmussenNature,2012_MeyerScience) \
188 WHERE Nr_SNPs >= 600000 \
189 "
190 .----- .-----
191 | Poseidon_ID | Nr_SNPs |
192 :=====:=====:
193 | Inuk.SG | 1101700 |
194 '-----'-----'

```

195 Ordering with ORDER BY

196 Order all individuals by Nr_SNPs.

```

197 $ qjanno " \
198 SELECT Poseidon_ID,Nr_SNPs \
199 FROM d(2010_RasmussenNature,2012_MeyerScience) \
200 ORDER BY Nr_SNPs \
201 "
202 .----- .-----
203 | Poseidon_ID | Nr_SNPs |
204 :=====:=====:

```

```

205 | A_French-4.DG          | 592535 |
206 | A_Ju_hoan_North-5.DG  | 593045 |
207 | A_Mbuti-5.DG          | 593057 |
208 | A_Dinka-4.DG          | 593076 |
209 | A_Yoruba-4.DG         | 593097 |
210 | A_Sardinian-4.DG      | 593109 |
211 | Inuk.SG               | 1101700 |
212 '-----'-----'

```

213 Order all individuals by Date_BC_AD_Median in a descending (DESC) order. Date_BC_AD_Median includes
 214 NULL values.

```

215 $ qjanno " \
216 SELECT Poseidon_ID,Date_BC_AD_Median \
217 FROM d(2010_RasmussenNature,2012_MeyerScience) \
218 ORDER BY Date_BC_AD_Median DESC \
219 "
220 .----- .----- .
221 | Poseidon_ID | Date_BC_AD_Median |
222 :=====:=====:
223 | Inuk.SG     | -1935             |
224 | A_Sardinian-4.DG |                 |
225 | A_Yoruba-4.DG |                 |
226 | A_Dinka-4.DG |                 |
227 | A_Mbuti-5.DG |                 |
228 | A_Ju_hoan_North-5.DG |             |
229 | A_French-4.DG |                 |
230 '-----'-----'

```

231 Reducing the number of return values with LIMIT

232 Only return the first three result individuals.

```

233 $ qjanno " \
234 SELECT Poseidon_ID,Group_Name \
235 FROM d(2010_RasmussenNature,2012_MeyerScience) \
236 LIMIT 3 \
237 "
238 .----- .----- .
239 | Poseidon_ID | Group_Name |
240 :=====:=====:
241 | Inuk.SG     | Greenland_Saqqaq.SG |
242 | A_Mbuti-5.DG | Ignore_Mbuti(discovery).DG |
243 | A_Yoruba-4.DG | Ignore_Yoruba(discovery).DG |
244 '-----'-----'

```

245 Combining tables with JOIN

246 For JOIN operations, SQLite requires table names to specify which columns are meant when combining multiple
 247 tables with overlapping column names. See the option -c/--showColumns to get the relevant table names as

248 generated from the input paths.

```
249 $ echo -e "Poseidon_ID,MoreInfo\nInuk.SG,5\nA_French-4.DG,3\n" > test.csv
```

250

```
251 $ qjanno "SELECT * FROM d(2010_RasmussenNature,2012_MeyerScience)" -c
```

```
252 .-----.-----.
```

```
253 |          Column          |          Path          |
```

```
254 :=====:
```

```
255 | Capture_Type             | d(2010_RasmussenNature,2012_MeyerScience) | ->
```

```
256 ...
```

```
257 -----.
```

```
258 | qjanno Table name       |
```

```
259 =====:
```

```
260 | d2010RasmussenNature2012MeyerScience |
```

```
261 ...
```

262

```
263 $ qjanno "SELECT * FROM test.csv" -c
```

```
264 .-----.-----.
```

```
265 | Column | Path | qjanno Table name |
```

```
266 :=====:
```

```
267 | Poseidon_ID | test.csv | test |
```

```
268 ...
```

269 Join the .janno files with the information in the test.csv file (by the Poseidon_ID column).

```
270 $ qjanno " \
```

```
271 SELECT d2010RasmussenNature2012MeyerScience.Poseidon_ID,Country,MoreInfo \
```

```
272 FROM d(2010_RasmussenNature,2012_MeyerScience) \
```

```
273 INNER JOIN test.csv \
```

```
274 ON d2010RasmussenNature2012MeyerScience.Poseidon_ID = test.Poseidon_ID \
```

```
275 "
```

```
276 .-----.-----.
```

```
277 | Poseidon_ID | Country | MoreInfo |
```

```
278 :=====:
```

```
279 | Inuk.SG      | Greenland | 5      |
```

```
280 | A_French-4.DG | France   | 3      |
```

```
281 '-----'-----'
```

282 Grouping data and applying aggregate functions

283 SQLite provides a number of aggregation functions: `avg(X)`, `count(*)`, `count(X)`, `group_concat(X)`,
284 `group_concat(X,Y)`, `max(X)`, `min(X)`, `sum(X)`. See the documentation [here](#). These functions can be well
285 combined with the `GROUP BY` operation.

286 Determine the minimal number of SNPs across all individuals.

```
287 $ qjanno "SELECT min(Nr_SNPs) AS n FROM d(2010_RasmussenNature,2012_MeyerScience)"
```

```
288 .-----.
```

```
289 | n |
```

```
290 :=====:
```

```

291 | 592535 |
292 '-----'

293 Count the number of individuals per Date_Type group and calculate the average Nr_SNPs for both groups.

294 $ qjanno " \
295 SELECT Date_Type,count(*),avg(Nr_SNPs) \
296 FROM d(2010_RasmussenNature,2012_MeyerScience) \
297 GROUP BY Date_Type \
298 "
299 .----- .----- .----- .
300 | Date_Type | count(*) | avg(Nr_SNPs) |
301 :===== :===== :===== :
302 | C14       | 1         | 1101700.0     |
303 | modern    | 6         | 592986.5      |
304 '-----'-----'-----'

```