Code Generation Instructions

The purpose of this document is to explain how to manipulate and use the code generation mechanism of the paper “Answering (Unions of) Conjunctive Queries using Random Access and Random-Order Enumeration”.

Example Query

To explain all details we will use a CQ as an example. The example query, denoted Q_{ex}, is defined as $Q_{ex}(x, y, z) \leftarrow R(x, y), S(y, z), T(z, v, w)$. We will also use the following join-tree:

$$\begin{tikzcd}
R(x,y) \arrow[r] & S(y,z) \arrow[r] & T(z,v,w)
\end{tikzcd}$$

Invocation

To invoke the generator, follow the following steps:

1. In the main directory prepare a subdirectory named “Qex”.
   Please note, the name of subdirectory <u>must</u> match the name that will later be written in the JSON files. Hence, please use the same name everywhere.
2. Fill out the 4 JSON files and place them in the query directory (see next section).
3. In the query directory create a subdirectory called “files”.
4. From the main directory run “python gen.py Qex”

JSON Files

Please note: it is mandatory that table names will not be prefixes of each other. For instance, having two tables named “R” and “R1” is not allowed. We highly recommend skirting the issue by adding underscores to the start and end of table names. Therefore, instead of “R”, “S”, “T” we will use “_R_”, “_S_”, “_T_”.

For further understanding, please refer to the filled example file under “example JSONs”.

IndexParams

This file contains details of the query and join-tree.

- <u>name</u>: the name of the query, that must match the name of the directory.
- <u>root</u>: the root of the join-tree.
- <u>projection</u>: a dictionary which details for every table whether it has projection.
- <u>postorder</u>: a list of the tables in root to leaf order.
- <u>preorder</u>: a list of the tables in leaf to root order.
- <u>tree</u>: the dictionary details the join-tree by detailing each the parent and children of each table (null as the parent of the root).

ParcelCreateList

We denote the fields of a relation as a “parcel”. This file defines all the parcels via the following fields:

- <u>name</u>: the name of the table this parcel belongs to

- <u>fields</u>: a list of the variables/fields. Each entry contains a type (the first element) and the name of the variable (the second element).
  Please note, as in the theoretic notation of CQs, all columns of the same name must be equal to each other. Meaning, if the table the x in _R_ should be equal to the x in _S_.
- <u>keyFields</u>: we denote by "key" the adhesion of the relation with its parent. Therefore, the key fields are those that the relation shares with its parent.
- <u>usecols</u>: this is a list of the fields in the *.tbl file that correspond to our columns (in case we don't want to use all columns, or sequential columns).
- <u>conversionFunctions</u>: a mapping between type to function that should be invoked in order to convert from a string to that type. The value given in the example could probably be left without change.
- <u>childrenInterAttr</u>: a dictionary that details the key fields of each child. Meaning, the fields this relation shares with each child. Please note, it must the fields given in the "keyFields" section of each child-relation.

ProjectedParcelCreateList

This file details all the different projections.

- <u>tbls</u>: The relations on which projection is performed. Please note, it must match the rest of the file as well as the "projection" dictionary in the file IndexParams.
- <u>array</u>: this is an array with an entry per projected relation. Each entry is similar to the parcel entry but only contains "name, fields, keyFields, childrenInterArr". All these components are defined the same way as previously, but they refer to the relation after projection.

Please note, if the files are compiled with –DPROJECTION_Qex (change according to query name), meaning that PROJECTION_Qex is defined for the preprocessor, than the projection will be performed. Otherwise, the projection will be ignored and the query will be computed as a full-join.

QueryFields

This file contains general details pertaining to the query.

- <u>name</u>: the name of the query. Please note, it must match the name in the file "IndexParams.json".
- <u>vars</u>: a list of all the variables in the query.
- <u>types</u>: a dictionary that maps each type to its type.
- <u>varsDict</u>: a dictionary that maps each relation to its variables.
- <u>freevarsDict</u>: a dictionary that maps each relation to it variables post-projection.
- <u>keyvarsDict</u>: a dictionary that maps each relation to the variables it shares with its parent.
- <u>freekeyvarsDict</u>: a dictionary that maps each relation to the variables it shares with its parent post-projection.

Adding Selection

Currently, we only support selection manually. To add selections go to the file "QexIndex.h". To the file, add a function with the signature:

$$bool\ Pred(void^*\ arg, void^*\ parcel)$$

The "arg" argument can be utilized however you see fit, while the parcel argument will contain the current row in the table (one of the parcel objects). The function should return "true" if the row is to be kept and "false" otherwise.

Please note, since UCQ algorithms compile different query enumerators, selection functions of different queries should have different names, or c++ will report a double definition error. For that sake, we recommend adding a prefix with the query name to all selection functions.

Using the Generated Files

To use the files generated perform the following steps:

1. in CQs/ run “mkdir Qex”
2. in CQs/Qex/ run “mkdir files”
3. in CQs/Qex/ run “mkdir TestsOutput”
4. Copy the generated Makefile, Qex_enumeration_test.cpp, and Q0_timing_test.cpp to Qex/
5. In CQs/Qex/ create a file “sampleratios.txt” and fill it with the different enumeration percentages that you would like, in a single row
6. Copy all the generated *.h files to “CQs/Qex/files/”

After these steps, you may use our experiment replication instructions to run the different experiments on the new query.