TikaEvalJdbc

Connecting tika-eval to your own db via JDBC

As default, tika-eval uses an in-memory H2 database, specified by `-db` on the commandline.

However, you can also specify a jdbc connection string with `-jdbc`.

NOTE: We will try to keep the SQL used in `Profile` and `Compare` simple enough to apply generally through jdbc. However, we cannot support all the flavors of SQL in our default `Report` tool. If you choose to use `-jdbc`, you are on your own for translating the report SQL to your dialect.

So far, I've only tested the jdbc connection with postgresql, derby in memory and H2 in memory. Please open a JIRA issue if you find problems with `Profile` or `Compare` with other flavors of SQL.

Examples

First, make sure to put the appropriate db driver jar on your classpath. In the following, we assume that you have the driver jar and the `tika-eval.jar` in `bin/`.

Basic Profile

```
    java -cp "bin/*" org.apache.tika.eval.TikaEvalCLI Profile -jdbc "jdbc:postgresql:tika_eval?
user=user&password=superSecret" -extracts extracts
```

Basic Profile with Custom Table Prefixes

Now, let's say you want to profile 3 separate runs of OCR with different settings, say `quality=1.0`, `quality=0.5` and `quality=0.1`. You'll need to prefix your tables with a different prefix for each run.

```
    java -cp "bin/*" org.apache.tika.eval.TikaEvalCLI Profile -jdbc "jdbc:postgresql:tika_eval?
user=user&password=superSecret" -extracts extracts10 -tablePrefix o10
```

```
    java -cp "bin/*" org.apache.tika.eval.TikaEvalCLI Profile -jdbc "jdbc:postgresql:tika_eval?
user=user&password=superSecret" -extracts extracts05 -tablePrefix o05
```

```
    java -cp "bin/*" org.apache.tika.eval.TikaEvalCLI Profile -jdbc "jdbc:postgresql:tika_eval?
user=user&password=superSecret" -extracts extracts01 -tablePrefix o01
```

This will leave you with 24 tables, 7 for each run and then 3 reference tables.

Then, let's say you want to compare the number of "common tokens" in each table for each container file. Unfortunately, the "ids" are unique *per run*, so you can really only focus on the container files, and you must join them by the file path.

So, step 1 is to build indices on the file path:

```
    create unique index path01_idx on o01_containers(file_path);
    create unique index path05_idx on o05_containers(file_path);
    create unique index path10_idx on o10_containers(file_path);
```

Then, join on the file_path to get the paths, and then add in the contents tables for each run:

```
select o10c.file_path,
    o10ct.num_common_tokens,
    o05ct.num_common_tokens,
    o01ct.num_common_tokens

from o10_containers o10c
    left join o05_containers o05c on o05c.file_path=o10c.file_path
    left join o01_containers o01c on o01c.file_path=o10c.file_path

    left join o10_contents o10ct on o10c.container_id=o10ct.id
    left join o05_contents o05ct on o05c.container_id=o05ct.id
    left join o01_contents o01ct on o01c.container_id=o01ct.id
```