Sphinx Module

MonetDB Extentions

Stefan de Konink

January 18, 2010

Open Data projects

- environmental data (RIVM, KNMI, VWA)
- business data (Chambre of Commerce, Tax)
- geospatial data (OpenStreetMap)

Technical setup

- Cherokee/DBSlayer
- MonetDB5/SQL (head)

Provides: Read-only connection by HTTP, using a SQL request and JSON, PHP, Ruby, Python, XML output.

http://dbslayer/SELECT * FROM tables;

Problems

- Businessdata contains 2.4 million rows
- Searching names, addresses, postalcodes
- All operations require caseinsensitive matching
- ILIKE cannot be avoided
- Materialised heuristic indices made to get performance surplus (current head shows the opposite)

Response from the community

- Lookup speed is too slow

Response from the community

- Lookup speed is too slow
- "Use another database!"

Response from the community

- Lookup speed is too slow
- "Use another database!"
- "Did you try Sphinx?"

Sphinx

- GPL licenced, (Free) Text Search, with **many** options
- Client/Server model
- Integration with MySQL
- Connectors for fetching from MySQL, PostgreSQL, ODBC, ...
- Query to DocumentID

Implementation in MonetDB

- License issues, thus by API, libsphinxclient (LGPL)
- KISS: Query + Index (for now)
- Map a DocumentID to a Key

Later if there is commercial interest:

- Query weighting
- Weighting results
- SphinxQL

Running Sphinx Queries from MonetDB5

- Install libsphinxclient
- Configure MonetDB5 (head) –with-sphinxclient
- –dbinit="include sphinx; include sql;"
- create function sphinx_searchIndex(query STRING, idx STRING) RETURNS TABLE (id bigint) EXTERNAL NAME sphinx."sphinx_searchIndex";
- select * from sphinx_searchIndex('MonetDB', 'openkvk');

Setting up Sphinx to index MonetDB5/SQL

- configure a source using type = odbc;
- odbc_dsn = Driver=MonetDB;Dbq=kvk;Uid=dbslayer; Pwd=dbslayer12345;Port=50001
- sql_query = select kvk, bedrijfsnaam, adres, postcode, plaats from kvk

Lookup time using current head, with cracker_pipe

We want to know the time to map a query to a KvK number.

- Sphinx lookup: 55ms
- ILIKE lookup: 646ms (cold), 571ms (hot)
- LIKE lookup: 127ms
- Equality lookup: 824ms (!)
- ILIKE (as exact): 435ms
- LIKE (as exact): 820ms (rewrite?)

...looking at the results, there is a fundamental issue to solve.