FishMark: A Linked Data Application Benchmark

Samantha Bail[1], Sandra Alkiviadous[1], Bijan Parsia[1], David Workman[2], Mark Van Harmelen[2], Rafael S. Goncalves[2], and Cristina Garilao[3]

[1] The University of Manchester, Manchester M13 9PL, United Kingdom
[2] HedTek, Manchester M4 1LZ, United Kingdom
[3] GEOMAR Helmholtz-Zentrum fr Ozeanforschung, 24148 Kiel, Germany

Abstract. FishBase is an important species data collection produced by the FishBase Information and Research Group Inc (FIN), a not-for-profit NGO with the aim of collecting comprehensive information (from the taxonomic to the ecological) about all the world's finned fish species. FishBase is exposed as a MySQL backed website (supporting a range of canned, although complex queries) and serves over 33 million hits per month. FishDelish is a transformation of FishBase into LinkedData weighing in at 1.38 billion triples. We have ported a substantial number of FishBase SQL queries to FishDelish SPARQL query which form the basis of a new linked data application benchmark (using our derivative of the Berlin SPARQL Benchmark harness). We use this benchmarking framework to compare the performance of the native MySQL application, the Virtuoso RDF triple store, and the Quest OBDA system on a fishbase.org like application.

1 Introduction

The Linked Open Data (LOD) movement promises much, indeed, nothing less than a new World Wide Web with comparable success to the Web as it is. The amount of LOD is growing at an interesting pace and the underlying technologies are constantly improving. Off the shelf, untuned RDF triple stores handle data sets on normal hardware that would have been unthinkable 5 years ago. However, there is scant evidence about the benefits and drawbacks of converting applications to use linked data. Given that the conversion from "native" data models (such as XML or relational databases) typically involves a large blow up in size and loss of tuned structures, e.g. indexes or (de)normalization, achieving comparable performance post-triplification is a common concern. While there may be other benefits to triplification, such as easier integration with other LOD, this needs to be weighed against the costs imposed.

To help assess the costs of triplification, we have developed FishMark, an application benchmark for linked data systems. FishMark consists of two components: The Manchester University Multi-Benchmarking (MUM-benchmark) framework, and a set of data, queries, and query frequences derived from FishBase, a comprehensive database about the world's finned fish species, and fish-

base.org, a popular web front end to FishBase.[4] FishBase is a LAMP application with a large number of SQL queries being invoked against a MySQL backend to generate various web pages. We have triplified FishBase and ported its most popular queries to SPARQL. FishMark thus allows a precise comparison between the RDBMS infrastructure and various linked data competitors.

Further, we have created an OWL ontology and respective mappings for the FishBase data, which allows us to measure the performance of an ontology-based data access (OBDA) system [6,4] compared to the RDMBS and RDF triple store versions. OBDA systems offer a different approach to the problem of describing domain knowledge at an abstract level while granting efficient and scalable access to large amounts of data. In the OBDA approach the data are held in external data sources, such as a standard relational database, which are connected to information in an OWL ontology using *mappings*. Ontologies used for this purpose are in the OWL 2 QL profile[5] which is based on the *DL-Lite* [1] family of description logics.

In this paper, we present a first approach to comparing the query performance of native RDBMS, RDF triple stores, and OBDA systems using a single set of—real—data and queries. In particular, we compare an Extract Transform Load (ETL) approach using D2R and the Virtuoso RDF store, and Quest, an OBDA system which executes the SPARQL queries via mappings against the original MySQL database.

2 Related Work

There have been a number of benchmarks for measuring the performance of SPARQL query answering, and countless approaches to SQL benchmarking; we therefore focus on a selection of the most prevalent RDF benchmarking frameworks.

One of the most well-known performance benchmarks for RDF stores is the Lehigh University Benchmark (LUBM) [5]. LUBM consists of a small, hand-built ontology containing information about university departments, students, etc., with a large number of instances for each class. The dataset is scaled by increasing the number of universities, which creates a randomised number of instances for the new university. This method generates entirely disjoint sets of data, a problem which the University Ontology Benchmark (UOBM) [7] seeks to rectify by generating *interrelations* between the instances across different universities.

The LUBM Benchmark is used in [11] as a benchmark for SparqlEngineDB, a SPARQL-to-SQL system which translates SPARQL queries to SQL queries and executes them against a relational database. This translation approach is thought to have an advantage over querying native RDF stores with SPARQL, as it does not require to hold the entire data in memory (as is the case with SPARQL), while also making use of the query optimisation techniques used in RDBMS.

[4] In general, we will use "FishBase" to refer to both parts.
[5] http://www.w3.org/TR/2008/WD-owl2-profiles-20081008/

The authors perform an evaluation of SparqlEngineDB against Virtuoso and Jena-SDB with the data scaling up to 200,000 triples. They find that Virtuoso exceeds the performance of both SparqlEngineDB and Jena-SDB.

The Berlin SPARQL Benchmark (BSBM) [2] uses a hand-built e-commerce dataset and a fixed set of queries to measure the performance of RDF triple stores. It provides a data generation tool which generates a data set of custom size, as well as a test driver framework for performing query mixes for several use cases, such as "explore", "update and explore", and "Business Intelligence", which emulate the *"search and navigation pattern of a consumer looking for a product"* [2]. The authors present the results of applying BSBM to measure the SPARQL query performance of several RDF triple stores, as well as the SQL translations of the SPARQL queries using Virtuoso RDF View and D2R Server.[6] While Sesame performs best for the smallest instance of the dataset (1 million items), Virtuoso's RDF View outperforms the triple stores on large-scale datasets. Even though no direct comparisons between the RDB and RDF stores' performance are made, it can be seen that the native SQL queries outperform the SPARQL queries by an order of magnitude on the smallest dataset.

One of the RDF benchmarks that use real test data and queries is the DBpedia SPARQL Benchmark (DBPSB) [8]. DBPSB uses data from the DBpedia[7] knowledge base and queries extracted from DBpedia's query logs. While implementing standard benchmarking techniques such as clearing the cache and performing warm-up runs, DBPSB also provides a method for scaling the size of the benchmarking dataset. In [8] the authors use DBPSB to test the performance of four triple stores: Virtuoso,[8] Sesame,[9] BigOWLIM,[10] and Jena-TDB.[11] They find that Virtuoso is the fastest of the four triple stores, handling large amounts of data significantly better than the other systems.

A similar approach to BSBM is the Social Network Intelligence Benchmark (SIB) [3], which uses a schema of a social network similar to Facebook, and data dictionaries in order to generate artificial RDF data. The benchmark also includes several query mixes, such as the "Interactive query mix" of 20 queries that simulate typical user interaction in a social network. The SP^2Bench benchmark [10] is set in a scenario similar to DBLP, a large computer science bibliography database which indexes over 200,000 citations. SP^2Bench includes a data generator to generate "large DBLP-like models" in RDF, which is based on a study of the features of DBLP and 12 hand-built SPARQL queries which vary in their characteristics.

[6] http://d2rq.org/d2r-server
[7] http://dbpedia.org/
[8] http://virtuoso.openlinksw.com/
[9] http://www.openrdf.org/index.jsp
[10] http://www.ontotext.com/owlim/editions
[11] http://jena.apache.org/documentation/tdb/index.html

3 Materials and Methods

3.1 Data

There are currently two FishMark data sets: A MySQL database representing a snapshot of FishBase from 2011, and an RDF graph that is the result of applying a D2R conversion to that database.

The conversion of the complete FishBase dump via D2R consumed several hours and resulted in an RDF graph with 1.38 billion triples (which was stored as a 250GB file). Initial tests with various triple stores, however, were unsuccessful when attempting to load the data. In order to deal with this issue, we generated another MySQL dump of FishBase which only included the tables needed for the given queries. This reduced the data resulting from the D2R conversion to approximately 20 million (20,186,776) triples. According to the Virtuoso statistics generator, this data set contains 31,927 fish species.

The FishBase OWL ontology which was manually created using the Protégé 4 ontology editor contains 10 classes, 10 object properties, 84 data properties, and 206 logical axioms. The manually created OBDA model contains 20 mappings, which map data from the RDB to the OWL ontology.

3.2 Queries and Query Mix

While, eventually, the FishMark query set should include at least all queries which drive FishBase.org, currently we have a select set of 22 SQL queries with a range of complexity and corresponding SPARQL translations. Short descriptions of all 22 queries are listed in Table 2 alongside the number of joins for each query; sample instances of a selected set of queries can be found in Appendix B. The queries are of restricted complexity due to the limited number of SPARQL features supported by the Quest OBDA system at the time of performing the benchmark.

We obtained a server log from the fishbase.org server in Kiel, Germany, for June 2012. The logs indicated that, on average, only a small number of distinct queries were performed on FishBase, the most frequent being the generation of a species page for a fish (2440 queries per day), followed by the search for a fish species by common name (1034 queries per day). The total numbers in June 2012, as well as the average daily and hourly frequency of the most frequently used queries (according to the FishBase server logs) are given in Table 3.

We use the following notation to describe aspects of the benchmarking framework:

- Query type: A parameterised named query, e.g. “Query 1: CommonName”, “Query 2: SpeciesPage”, etc.
- Query instance: An instance of a query type with randomly selected values for the parameters.
- Query set: A set of query instances of all query types in the query mix.

We have defined three distinct benchmarks which are intended to test various aspects of the systems:

ID	Query name	Description	Joins
1	CommonName	Find species for a given common name.	2
2	SpeciesPage	Find information about a specific species.	5
3	Genus	Find species matching a given genus.	1
4	Species	Find species matching a given species.	1
5	FamilyInformation	Find information about a family of species.	1
6	FamilyAllFish	Find all fish for a given family.	1
7	FamilyNominalSpecies	Find all nominal species for a given family.	1
8	FamilyListOfPictures	Find all pictures for a given family.	2
9	CollaboratorPage	Retrieve information about a collaborator.	0
10	PicturePage	Retrieve information about a picture.	1
11	CAllFish	Find all fish for a given country.	3
12	CSpeciesInformation	Find information about a species of a country.	2
13	CFreshwater	Find all freshwater species for a country.	3
14	CIntroduced	Find all introduced species for a country.	3
15	CEndemic	Find all endemic species for a country.	3
16	CReefAssociated	Find all reef-associated species for a country.	3
17	CPelagic	Find all pelagic species for a country.	3
18	CGameFish	Find all game fish for a country.	2
19	CCommercial	Find all commercial fish for a country.	4
20	CUsedAquaculture	Find all species used for a. c. for a country.	3
21	CPotAquaculture	Find species w/ potential use for a.c. for a country.	2
22	CAquariumTrade	Find all species used for a. t. for a country.	3

Table 2. FishBase Queries with short descriptions and number of joins.

Benchmark 1: Individual Queries The first query mix for the benchmark consists of a simple performance test for each individual query, i.e. the query mix is used as a simple means to run each query exactly once. We generated multiple query sets for this test, each with a new, randomly selected set of parameters. Generating multiple query sets seemed necessary, as the data in FishBase are fairly heterogeneous: For example, a common name search for "Shark" returns 24 different species, while the same search for "Borna Snakehead" returns exactly 1 result. Therefore, running the query mix with the same query set may skew the results towards "good" or "bad" parameters for the queries. The Berlin SPARQL Benchmark employs the same principle to generate different parameter values for queries.

Benchmark 2: Randomised Weighted Query Mix A second variant of the query mix is the randomised weighted query mix based on the FishBase server access logs. The query mix contains the 5 most frequent query types, each of the queries being instantiated n times, where n is the frequency of the query according to the server access logs. The final query mix contains 175 query instances of 5 query types. Note that the queries are instantiated with random parameter values, therefore it is possible that some of the query instances are identical. This seems realistic, as FishBase users might perform the same query several times in a row.

Query name	Month	Day	Hour
Species Page	73213	2440.43	101.68
Common Name	31008	1033.60	43.07
Genus	13331	444.37	18.52
Country Species Information	4429	147.63	6.15
Collaborator Page	4138	137.93	5.75

Table 3. Frequency of the most common FishBase queries per month (total), day (mean), and hour (mean), June 2012.

Benchmark 3: Typical User Scenario The log files also allow us to draw conclusions as to how users commonly navigate on the fishbase.org site. As the species page for a fish species is the central point of information, the log files show a usage pattern which focuses heavily on accessing species pages from various other points on the site, most frequently the common name search (which is a prominent feature on the fishbase.org start page). From this usage pattern, we can construct a query mix which emulates the route a typical user takes during one visit to fishbase.org, similar to BSBM's *explore* use case.

3.3 Datastores- and Access

Virtuoso Open Source 6.1.5 Virtuoso is a "multi-model data server" which supports data storage and management of relational data, XML data, and RDF data, amongst others. Through several prior benchmarks, Virtuoso emerged as one of the best performing RDF triple stores. We installed Virtuoso (Open Source edition) following the instructions on the Virtuoso wiki[12] for a default install. As recommended by Virtuoso, the following parameters were modified in the virtuoso.ini file to match our hardware setup:

- NumberOfBuffers: 1360000
- MaxDirtyBuffers: 1000000
- MaxCheckpointRemap: 60GB (= 1/4 of the DB size)

These were the only measures taken to tune the datastore. The FishBase RDF triples were then loaded into the database from the n-triples file using the SQL command DB.DBA.TTLP_MT.

MySQL 5.5 Relational DBMS The current live version of FishBase uses a MySQL RDBMS as data store. As described above, we generated a smaller snapshot of the FishBase database dump which contained all the information required by the queries in the query mix. The data was loaded into an "out-of-the-box" install of the MySQL 5.5 RDBMS running on our test machine.

[12] http://www.openlinksw.com/dataspace/dav/wiki/Main

Quest 1.7 OBDA System (using a MySQL database) The Quest OBDA system [9] defines "virtual ABoxes" over data which is stored in a relational DBMS. It provides access to the data via SPARQL queries which the system rewrites into SQL, thereby delegating the query execution entirely to the RDBMS. Quest currently supports several RDBMS, including PostgreSQL, MySQL, DB2 and Oracle. For our benchmark, we used the existing MySQL database with the manually created OWL ontology and mappings described above.

3.4 Benchmarking Framework

We developed a multi-query language benchmarking framework which is a derivative of the Berlin SPARQL Benchmark (BSBM) code, extending its functionality by a query generation module and additional connectors for SQL queries and OBDA access. The Manchester University Multi-Benchmarking (MUM-benchmark) framework is released as an open source project under the Apache License 2.0.[13]

The query generator accepts an XML file containing parameterised queries (in SPARQL or SQL) and the respective queries to select a random parameter value from the data store for each parameter. This allows us to generate queries based on data from existing data stores, rather than relying on artificially generated data and queries. Examples of such a parameterised query can be found in Appendix A.

The BSBM TestDriver has been extended to measure the performance of relational databases queries via SQL queries and OBDA. The benchmark TestDriver measures the query performance of a *query mix*, which is created manually by specifying a series of queries in a plain text file. The benchmarking process includes a warm-up phase of several runs (default: 50 runs) of the query mix, which is followed by multiple runs (default: 500 runs) whose performance is measured. The results of the benchmark are then output as an XML result file, including aggregated metrics for the query mix, as well as individual metrics for each query in the query mix.

3.5 What We Measure

Due to time constraints, we focused on obtaining results for Benchmark 1 described in section 3.2, which measures the performance of each individual query. We generated 20 distinct query sets in order to ensure that the query performance was not affected by the choice of parameters. Each of the query mixes in the benchmark was run 50 times as a warm-up phase, followed by 100 timed runs;[14] this results in a total of 2,000 measured runs per query.

[13] http://code.google.com/p/mum-benchmark/

[14] Note that the default number of timed runs in the BSBM framework is 500; due to the large number of query sets and the relatively stable times after the warm-up, we reduced the number of timed runs to 100.

In preliminary tests we found that the Quest system performed reasonably well on the query mix, with one significant drawback: The query rewriting consumed a large amount of time for some of the queries, in particular for query 2 (SpeciesPage). This is due to the large number of SQL queries being generated in the SPARQL-to-SQL rewriting process, which then have to be pruned in a time-consuming process. In the case of query 2, the rewriting algorithm generated 7,200 queries, of which 7,198 were pruned, whereas most other queries in the mix generated not more than 120 queries. The time required to rewrite query 2 was nearly 10 seconds on average, which caused extreme delays in the benchmark. Due to a simple caching mechanism, however, the system did not have to perform any rewriting after the first execution of the query, which lead to a significant improvement in the query execution time. We therefore performed the rewriting and caching in the warm-up phase of the benchmark, only measuring the execution time of the SQL query generated by Quest. While this may seem to paint an unrealistic picture of the total execution time of the queries, it does provide us with a measure of the quality of the SPARQL-to-SQL rewriting.

The results are returned using an extended version of the BSBM result XML file, which includes metrics for the query mix (fastest, slowest, and average query mix run time, query mixes per hour), and metrics for the individual queries: Average / min / max query execution time, queries per second, average / min / max number of results per query execution, and number of query timeouts.

3.6 Hardware

The data stores were installed on an "out-of-the-box" Mac Mini with a 2.7 GHz Intel Core i7 dual-core processor, 16 GB (1333 MHz DDR3) memory, and a 750 GB HDD, running Mac OS X 10.7.4 (Lion). The benchmark queries were performed remotely, using an identical machine.

4 Results

The results of the individual query benchmark are shown in Table 4, the unit being queries per second (qps).[15] The SQL query results are given only as a baseline to compare the other datastores against. As stated previously, the results for Quest do not exclude the query re-writing times, but only compare the performance of the SQL queries the OBDA system generates from the SPARQL queries. The SPARQL queries against Virtuoso perform consistently worse than against Quest, with Quest outperforming Virtuoso by roughly an order of magnitude on most queries.

Across all 22 queries, there are large differences in performance for each data store: For 7 of the 22 queries, Virtuoso's performance lies in the single-digit range, a further 10 only range between 12 and 68, and only 5 queries (CommonName,

[15] Please note that query 9 was wrongly generated for Quest, therefore we did not obtain any meaningful results for this query.

FamilyAllfish, FamilyNominalSpecies, Genus, Species) perform better than 100 qps, with the best query execution time for query 4 (Species). The performance even drops to only 1 query per second on average for query 2 (SpeciesPage).

ID	Query name	Virtuoso	Quest	MySQL
1	CommonName	132	850	1262
2	SpeciesPage	1	552	840
3	Genus	167	753	1025
4	Species	192	870	1249
5	FamilyInformation	17	572	840
6	FamilyAllfish	141	704	1113
7	FamilyNominalSpecies	161	773	1060
8	FamilyListOfPictures	50	578	742
9	CollaboratorPage	27		824
10	PicturePage	68	711	1166
11	CAllFish	24	316	629
12	CSpeciesInformation	7	598	1009
13	CFreshwater	6	201	212
14	CIntroduced	9	303	479
15	CEndemic	13	656	985
16	CReefAssociated	4	266	378
17	CPelagic	7	337	532
18	CGameFish	9	580	845
19	CCommercial	12	556	748
20	CUsedForAquaculture	12	779	1229
21	CPotentialAquaculture	34	694	957
22	CAquariumTrade	66	815	1251

Table 4. Query results of the 22 queries in qps (queries per second)

The behaviour of Quest seems more stable compared to Virtuoso: 16 of the 22 queries perform at between 550 and 870 qps, with only query 13 (Country Freshwater, i.e. retrieve all freshwater species of a given country, an instantiation of which is shown in Appendix B) and 16 (Country Reef Associated) causing a significant drop in performance across all three systems. This is surprising, as the Country queries (query 11 to 22) are all similar, but rank among the best and worst performing queries for the Quest system. The search for a Species (query 4) as well as the Common Name query (query 1) are executed at the highest speed, with 870 and 850 qps, respectively.

In summary, it can be said that the SQL queries generated by Quest perform surprisingly well compared to the Virtuoso RDF store. As Virtuoso has consistently performed well in existing benchmarks (see Section 2), we have reason to assume that the comparatively weak performance is not restricted to Virtuoso, but rather a general issue of triple stores. While the results of the native SQL queries against the MySQL database are only given as a baseline, it is clear to

see that the results of the queries against Quest come fairly close to those of the native SQL queries.

5 Conclusions and Future Work

In this paper, we have presented a multi-purpose benchmarking framework, which allows benchmark users to generate randomised queries from existing data. We have used this framework to generate a benchmark using data from FishBase, a large collection of information about the world's fish. The *FishMark* has been employed to measure the performance of several data stores. While the work on this benchmark is in its early stages, we have found that the combination of real data, query mixes inferred from server logs, automated query instantiation and query benchmarking over several different systems (RDF triple store, SQL RDBMS, OBDA using an RDBMS), makes for a promising approach to performance measurement. The OBDA system we tested outperformed the RDF store by approximately an order of magnitude, while being surprisingly close in performance to FishBase's native relational database.

For future work, there are a number of possible ways to extend and improve the MUM-benchmarking framework. The current version only generates independent parameter values for the queries in a query mix rather than a *sequence* of queries in which each parameter value depends on the parameter values in previous queries. Sequencing would generate a more natural set of queries for a query mix.

We are planning to develop strategies for scaling the FishBase data based on a single fish species. This will allow us to test the performance of various data engines using self-contained subsets of the FishBase data. Another next step is the generation of more realistic query mixes based on the information extracted from the FishBase server logs. Additionally, we are aiming to make another attempt at using the complete FishBase data set (1.38 billion triples) for the benchmark.

The main purpose of the benchmarking results in this report is to demonstrate the FishMark benchmark; therefore we only tested a very restricted number of systems on a single query mix. We aim to perform more extensive tests with FishMark using a larger set of RDF stores, SPARQL-to-SQL rewriters, and OBDA systems. Finally, we did not attempt to measure the query performance under the load of multiple clients, which is a natural next step in the development of the FishMark benchmark.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. of Artificial Intelligence Research 36, 1–69 (2009)
2. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semantic Web Inf. Syst. 5(2), 1–24 (2009)

3. Boncz, P., Pham, M.D., Erling, O., Mikhailov, I., Rankka, Y.: Social network intelligence benchmark (SIB) - version 0.8. http://www.w3.org/wiki/Social_Network_Intelligence_BenchMark (2011)
4. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web J. 2(1), 43–53 (2011)
5. Guo, Y., Pan, J.Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. of Web Semantics 3(2-3), 158–182 (2005)
6. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Proc. of IJCAI-11. pp. 2656–2661 (2011)
7. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Proc. of ESWC-06. pp. 125–139 (2006)
8. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.C.N.: DBpedia SPARQL benchmark - performance assessment with real queries on real data. In: Proc. of ISWC-11. pp. 454–469 (2011)
9. Rodriguez-Muro, M., Calvanese, D.: Quest, an OWL 2 QL reasoner for ontology-based data access. In: Proc. of OWLED-12 (2012)
10. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL performance benchmark. In: Proc. of ICDE-09. pp. 222–233 (2009)
11. Weiske, C., Auer, S.: Implementing SPARQL support for relational databases and possible enhancements. In: Proc. of CSSW-07. pp. 69–80 (2007)

Appendix A: Sample Query Template

Query Template for Query 2: SpeciesPage (SPARQL)

```
<bmquery name="SpeciesPage">
     <query>
         <![CDATA[
             PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
             PREFIX fish: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
             SELECT ?common ?code ?refno
             ?author  ?demerspelag ?anacat
             ?family ?order ?class ?entered
             ?pic ?picid ?description ?refauthor ?refyear
             ?collaborator  ?comments
             WHERE {
             ?x fish:species_Genus "%genus%" .
             ?x fish:species_SpecCode ?code.
             ?x fish:species_Species "%species%" .
             ?x fish:species_Comments ?comments .
             OPTIONAL {?x fish:species_Author ?author}.
             OPTIONAL {?x fish:species_FBname ?common}.
             OPTIONAL {?x fish:species_SpeciesRefNo ?refno}.
             OPTIONAL {?ref fish:refrens_RefNo ?refno}.
             OPTIONAL {?ref fish:refrens_Author ?refauthor}.
             OPTIONAL {?ref fish:refrens_Year ?refyear}.
             OPTIONAL {?x fish:species_Comments ?biology.}
             OPTIONAL {
             ?x fish:species_FamCode ?famcode.
             ?famcode fish:families_Family ?family.
             ?famcode fish:families_Order ?order.
             ?famcode fish:families_Class ?class.
             }
             OPTIONAL {?morph fish:morphdat_Speccode ?x.
             ?morph fish:morphdat_AddChars ?description.}
             OPTIONAL {?x fish:species_DemersPelag ?demerspelag.}
             OPTIONAL {?x fish:species_AnaCat ?anacat.}
             OPTIONAL {?x fish:species_PicPreferredName ?pic.
             ?pic_node fish:picturesmain_SpecCode ?x.
             ?pic_node fish:picturesmain_PicName ?pic.
             ?pic_node fish:picturesmain_autoctr ?picid.
             ?pic_node fish:picturesmain_Entered ?entered.
             ?pic_node fish:picturesmain_AuthName ?collaborator.
             }
             }
         ]]>
     </query>
     <parameterquery>
       <paramname>genus</paramname>
       <paramname>species</paramname>
         <paramvalues>
             <query>
                 <![CDATA[
                     PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
                     SELECT ?genus ?species
                     WHERE {
                     ?code fd:species_Genus ?genus .
                     ?code fd:species_Species ?species .
                     }
                 ]]>
             </query>
         </paramvalues>
     </parameterquery>
 </bmquery>
```

Appendix B: Sample Benchmark Queries

Instance of Query 1: Common Name (SPARQL)

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?type ?species ?genus ?country ?language
WHERE {
?nameID fd:comnames_ComName "Banded wormfish" .
?nameID fd:comnames_NameType ?type .
?nameID fd:comnames_SpecCode ?code .
?nameID fd:comnames_C_Code ?ccode .
?code fd:species_Species ?species .
?code fd:species_Genus ?genus .
?ccode fd:countref_PAESE ?country .
}
```

Instance of Query 2: SpeciesPage (SPARQL)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fish: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?common ?code ?refno
?author  ?demerspelag ?anacat
?family ?order ?class ?entered
?pic ?picid ?description ?refauthor ?refyear
?collaborator  ?comments
WHERE {
?x fish:species_Genus "Sebastes" .
?x fish:species_SpecCode ?code.
?x fish:species_Species "nigrocinctus" .
?x fish:species_Comments ?comments .
OPTIONAL {?x fish:species_Author ?author}.
OPTIONAL {?x fish:species_FBname ?common}.
OPTIONAL {?x fish:species_SpeciesRefNo ?refno}.
OPTIONAL {?ref fish:refrens_RefNo ?refno}.
OPTIONAL {?ref fish:refrens_Author ?refauthor}.
OPTIONAL {?ref fish:refrens_Year ?refyear}.
OPTIONAL {?x fish:species_Comments ?biology.}
OPTIONAL {
?x fish:species_FamCode ?famcode.
?famcode fish:families_Family ?family.
?famcode fish:families_Order ?order.
?famcode fish:families_Class ?class.
}
OPTIONAL {?morph fish:morphdat_Speccode ?x.
?morph fish:morphdat_AddChars ?description.}
OPTIONAL {?x fish:species_DemersPelag ?demerspelag.}
OPTIONAL {?x fish:species_AnaCat ?anacat.}
OPTIONAL {?x fish:species_PicPreferredName ?pic.
?pic_node fish:picturesmain_SpecCode ?x.
?pic_node fish:picturesmain_PicName ?pic.
?pic_node fish:picturesmain_autoctr ?picid.
?pic_node fish:picturesmain_Entered ?entered.
?pic_node fish:picturesmain_AuthName ?collaborator.
}
}
```

Instance of Query 3: Genus

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?species ?author ?family ?ename
WHERE {
?code fd:species_Species ?species .
?code fd:species_Genus "Parachondrostoma" .
OPTIONAL {?code fd:species_FBname ?ename .}
?code fd:species_Author ?author .
?code fd:species_FamCode ?fcode .
?fcode fd:families_Family ?family .
}
```

Instance of Query 4: Species

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?genus ?author ?family ?ename
WHERE {
?code fd:species_Species "ocellatum" .
?code fd:species_Genus ?genus .
OPTIONAL {?code fd:species_FBname ?ename .}
?code fd:species_Author ?author .
?code fd:species_FamCode ?fcode .
?fcode fd:families_Family ?family .
}
```

Instance of Query 5: Family Information

```
 PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?order ?class ?noOfGenera ?noOfSpecies ?marine ?brackish ?freshwater ?fossil
?aquarium ?remark  ?division ?activityLevel ?author ?year  ?repguild  ?SpeciesCount
WHERE {
 ?familiesID fd:families_Family "Ipnopidae" .
 ?familiesID fd:families_Order ?order .
 ?familiesID fd:families_Class ?class .
 ?familiesID fd:families_SpeciesCount ?SpeciesCount .
 ?familiesID fd:families_Genera ?noOfGenera .
 ?familiesID fd:families_Species ?noOfSpecies .
 OPTIONAL {?familiesID fd:fossilReference ?fossil }.
 ?familiesID fd:families_Marine ?marine .
 ?familiesID fd:families_Brackish ?brackish .
 ?familiesID fd:families_Freshwater ?freshwater .
 ?familiesID fd:families_Aquarium ?aquarium .
 ?familiesID fd:families_Remark ?remark .
 ?familiesID fd:families_Remark ?remark .
 ?familiesID fd:families_Division ?division .
 ?familiesID fd:families_Activity ?activityLevel .
 ?familiesID fd:families_ReprGuild ?repguild .
 ?familiesID fd:families_FamiliesRefNo ?code .
 ?x fd:refrens_RefNo ?code .
 ?x fd:refrens_Author  ?author .
 ?x fd:refrens_Year ?year .
}
```

Instance of Query 6: Family All Fish

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?species ?genus ?author ?englishName
WHERE { ?SpeciesID fd:species_Author ?author ;
fd:species_Species ?species;
fd:species_Genus ?genus ;
fd:species_FamCode ?code .
?code  fd:families_Family "Stromateidae" .
OPTIONAL {?SpeciesID fd:species_FBname ?englishName } .
}
```

Instance of Query 7: Family Nominal Species

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?species ?author ?genus ?ref
WHERE { ?SpeciesID fd:species_Author ?author ;
fd:species_Species ?species;
fd:species_Genus ?genus ;
fd:species_FamCode ?code .
OPTIONAL {?SpeciesID fd:species_ImportanceRef ?ref }.
?code  fd:families_Family "Gobiesocidae" .
}
```

Instance of Query 8: Family List of Pictures

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?genus ?species ?englishname ?picture ?photographer ?location
WHERE {?picID fd:picturesmain_SpecCode ?code ;
fd:picturesmain_PicName ?picture ;
fd:picturesmain_AuthName ?photographer .
OPTIONAL { ?picID fd:picturesmain_Locality ?location }.
OPTIONAL { ?code fd:species_FBname ?englishname } .
?code fd:species_Species ?specie;
fd:species_Genus ?genus ;
fd:species_FamCode ?fcode .
?fcode fd:families_Family "Moronidae" .
}
```

Instance of Query 9: Collaborator Page

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?prename ?surname  ?email ?photo ?webpage ?fax ?institute ?street ?city ?country ?comments ?keywords ?year
WHERE {
?x fd:collaborators_Personnel "1952" .
OPTIONAL {?x fd:collaborators_Prename ?prename }.
OPTIONAL {?x fd:collaborators_Surname ?surname }.
OPTIONAL {?x fd:collaborators_E-mail ?email }.
OPTIONAL {?x fd:collaborators_StaffPhoto ?photo }.
OPTIONAL {?x fd:collaborators_WebPage ?webpage }.
OPTIONAL {?x fd:collaborators_FAX ?fax }.
OPTIONAL {?x fd:collaborators_Institute ?institute }.
```

```
OPTIONAL {?x fd:collaborators_Street ?street }.
OPTIONAL {?x fd:collaborators_City ?city }.
OPTIONAL {?x fd:collaborators_Country ?country }.
OPTIONAL {?x fd:collaborators_Comments ?comments }.
OPTIONAL {?x fd:collaborators_Keywords ?keywords }.
OPTIONAL {?x fd:collaborators_Year ?year }.
}
```

Instance of Query 10: Picture Page

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?genus ?species ?photographer ?size ?location ?stage  ?reference ?remark
WHERE {
?pcode fd:picturesmain_PicName "Danav_u0.jpg" .
?pcode fd:picturesmain_AuthName ?photographer .
OPTIONAL {?pcode fd:picturesmain_Size ?size }.
OPTIONAL {?pcode fd:picturesmain_Locality ?location }.
?pcode fd:picturesmain_LifeStage ?stage .
OPTIONAL {?pcode fd:picturesmain_Reference ?reference }.
OPTIONAL {?pcode fd:picturesmain_Remark ?remark }.
?pcode fd:picturesmain_SpecCode ?scode .
?scode fd:species_Genus ?genus .
?scode fd:species_Species ?species .
}
```

Instance of Query 10: Family All Fish

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?species ?genus ?author ?englishName
WHERE { ?SpeciesID fd:species_Author ?author ;
fd:species_Species ?species;
fd:species_Genus ?genus ;
fd:species_FamCode ?code .
?code  fd:families_Family "Stromateidae" .
OPTIONAL {?SpeciesID fd:species_FBname ?englishName } .
}
```

Instance of Query 13: Country Freshwater

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?order ?family ?genus ?species ?occurrence ?fbname ?name
WHERE {
?nameID fd:comnames_ComName ?name .
?nameID fd:comnames_C_Code ?ccode .
?nameID fd:comnames_SpecCode ?x.
?x fd:species_Genus ?genus .
?x fd:species_Species ?species .
OPTIONAL {?x fd:species_FBname ?fbname }.
?x fd:species_FamCode ?f .
?f fd:families_Family  ?family .
?f fd:families_Order ?order .
?c fd:country_SpecCode ?x.
?c fd:country_Status ?occurrence .
?c fd:country_Freshwater 1 .
?c fd:country_C_Code ?cf .
?cf fd:countref_PAESE "Ghana".
}
```

Instance of Query 16: Country Reef Associated

```
PREFIX fd: <http://fishdelish.cs.man.ac.uk/rdf/vocab/resource/>
SELECT ?order ?family ?genus ?species ?occurrence ?fbname ?name ?dangerous
WHERE {
?nameID fd:comnames_ComName ?name .
?nameID fd:comnames_C_Code ?ccode .
?nameID fd:comnames_SpecCode ?x.
?x fd:species_Genus ?genus .
?x fd:species_Species ?species .
?x fd:species_Dangerous ?dangerous .
?x fd:species_DemersPelag "reef-associated" .
OPTIONAL {?x fd:species_FBname ?fbname }.
?x fd:species_FamCode ?f .
?f fd:families_Family  ?family .
?f  fd:families_Order ?order .
?c fd:country_SpecCode ?x.
?c fd:country_Status ?occurrence .
?c fd:country_C_Code ?cf .
?cf fd:countref_PAESE "Trinidad Tob" .
}
```