COMP62421
Querying Data on the Web
Coursework Manual

Department of Computer Science
University of Manchester

On Assigned Readings

These papers provide background information of relevance to the material in the unit. In addition, the contents of these papers may be examined in the quizzes.

Week 1

Surajit Chaudhuri: **An Overview of Query Optimization in Relational Systems**. PODS 1998: 34-43 http://doi.acm.org/10.1145/275487.275492

Week 2

Craig Brown, Xavier Franc, Michael Paddon, **Native XML Databases: Death or Coming of Age?** XMLPrague 2015 Conference Proceedings, pp. 107-119. http://archive.xmlprague.cz/2015/files/xmlprague-2015-proceedings.pdf [pp. 107-119 only]

Week 3

Olaf Hartig, Andreas Langegger: **A Database Perspective on Consuming Linked Data on the Web**. Datenbank-Spektrum 10(2): 57-66 (2010). http://dx.doi.org/10.1007/s13222-010-0021-7

Week 4

Jeffrey D. Ullman: **Designing Good MapReduce Algorithms**. ACM Crossroads 19(1): 30-34 (2012) http://doi.acm.org/10.1145/2331042.2331053

On Coursework

There are two types of coursework in COMP62421: quizzes and lab work.

On Quizzes

- Quizzes are done synchronously, and as a result are timetabled.
- The topics for a quiz on a given teaching week consist of the material in the assigned articles for reading in the previous weeks, as well as all the taught content thus far in the course unit (and therefore not just in that specific teaching week).
- There are 4 quizzes, which run in Weeks 2 to 5.

On Duration, Question Types, and Consultation

- Quizzes last 25 minutes.
- They consist of five multiple choice style questions.
- Incorrect answers do not subtract from the total.

On Lab Work

- Lab work is associated with timetabled lab sessions.
- The topics for lab work consist of application and exploration of the taught content in the course unit.
- All the lab work can be carried out on departmental computers or on your own machines, though the descriptions are generally targeted at and have been tested on unix.
- The lab machines have sqlite3 installed; if you want to do the labs that use sqlite3 (RW, QO) on a laptop, you may have to install SQLite (https://www.sqlite.org/download.html).
- The software used in the unit uses licenses that will be suitable for public use, so you can download and install software to your personal machine. However, be aware that we may struggle to support students with machine-specific issues.

On Lab Work's Contribution to the Course Unit Mark

- There are three pieces of lab work, RQ (Week 1), QO (Weeks 2-3) and SP (Weeks 4-6).
- The coursework mark is subdivided as follows:
    - Quizzes: 10%
    - Lab RQ: 0% (so this is a *formative* assessment – you will be given the solutions).
    - Lab QO: 40%
    - Lab SP: 50%
- The coursework marks, as a whole, contribute 50/100 marks towards the final course unit mark.
- The remaining 50/100 comes from the exam.

On Deadlines

The deadlines on the lab work are as follows:

- QO: Week 4, Thursday at 18:00
- SP: Week 6, Friday at 18:00

On What Your Submission Consists of and Where to Upload it

The submission is always a report in PDF format. This must be such that we can open it and read it to mark it. Anything that prevents us from doing so cannot be compensated. You should upload the report associated with each submission as a pdf document called:

```
<username>_COMP62421_LW_Report
```

On Preparing a Report File

We expect you to format your report in a scholarly manner. For example, formal language expressions and code should be properly indented and use monospaced (i.e., fixed-width) fonts — such as `Monaco`, `Andale Mono`, `Courier`, etc. — and mathematical expressions should be clearly and properly written down.

In your comments, analyses, etc., be brief and to the point. There is never any need to write an essay on anything in this course unit: at most, all we are looking for is technical comments and technical arguments.

In all cases, if you believe that relevant information is missing, then use comments in your submission to make explicit any assumptions that were required for you to answer.

If your assumptions are convincing (i.e., if they are *both* made in response to genuine uncertainty and incompleteness *and* are consistent with all the information explicitly given), we will take them into account in the marking.

On Plagiarism and Working Together

You must always work individually. You must only consult and discuss things with your friends once you have gained a thorough understanding of what constitutes academic malpractice. You are not allowed to copy solutions from anyone else, or to pass solutions to anyone else, in any form (conversation, paper, electronic media, etc.), unless otherwise authorized explicitly. We will run plagiarism detection software on submissions.

Further guidance on plagiarism and cheating is available here:

`https://documents.manchester.ac.uk/display.aspx?DocID=2870`

Lab Work RQ

There is material provided on Blackboard to support the execution of SQL (using SQLite) and Relational Algebra (using RA). All queries are to be written over the Mondial database, which is provided in the download.

Task 1: Write *tuple-relational calculus (TRC) expressions* that, upon evaluation, return the data characterized by each of the following English-language specifications:

(1) Return the name of any country that has a lake.

(2) Return all the available attributes on cities whose population is between 3 and 5 million inhabitants.

(3) Return the country code and the continent of every country not in Europe or in Australia/Oceania.

(4) Return the names of countries that also give their name to one of its own provinces.

(5) Return the names of countries that are not landlocked (i.e., have a sea coast).

Task 2: Write *relational-algebraic (RA) expressions* that, upon evaluation, return the data characterized by each of the following English-language specifications.

(6) Return the names of countries that are not landlocked (i.e., have a sea coast).

(7) Return the names of all lakes, rivers and seas.

(8) Return the name of the country, and the names of the organizations of which the country is a member, for countries with Buddhist populations.

(9) Return the names of countries that also give their name to one of its own provinces.

(10) Return, for every river in the United Kingdom, the length of that river.

(11) Return the names of all cities that have a population larger than that of the capital city of the country.

Task 3: Write *SQL expressions* that, upon evaluation, returns the data characterized by each of the following English-language specifications. Use duplicate removal where appropriate (e.g., when a duplicate is not required in the intended answer).

(12) Return the names of up to 10 countries and the value corresponding to half the country's population.

(13) Return all the information available in the City table about cities whose name is Manchester.

(14) Return the name of cities whose name starts with the substring 'Man'.

(15) Return the name of the country, and the names of the organizations of which the country is a member, for countries with Buddhist populations and organizations established after 1st December 1994.

(16) Return the name of each country with the number of islands in it.

Some Comments

In all the queries above, do not get bogged down by the complexity of the Mondial encoding of geographic properties. Mondial is not a spatial database and lacks spatial types (and corresponding spatial operations).

For Task 1, you are expected to submit the TRC expressions only. You are not required to run the query against the data as there is no easily available TRC evaluator to use, but if you want to ensure your expression computes the intended result, consider mapping the expression into an iterative computation in your favourite language.

For Task 2, you should use the **RA** software[1] to evaluate your expressions. We propose using the command line options to read the input from a file and to redirect the output to a file, as described in the download that is available on Blackboard. Note that **RA** is dependent on the **SQLite** DBMS.

For Task 3, you should use **SQLite** to evaluate your expressions.

Software/Data

A folder is provided in Blackboard that gives you access to relevant data and scripts.

On Scripting, Echoing and Logging

We note that familiarity with **SQLite** will be useful for Coursework QO. To give an example using **SQLite**, let's suppose that the query to be evaluated is:

Return all the information about any 4 countries

The answer to this is:

```
select * from country limit 4;
```

You would then compose a script, let's call it `Atan_Luring_LW1.sql` (note the suffix), that, in this simple case, would be as follows:

```
-- Atan_Luring_LW1.sql
--

-- set echoing on
.echo on

-- set spooling out: note the suffix
.output Atan_Luring_LW1.log

--
select * from country limit 4;
```

You can then start **SQLite** (using the `sqliterc` initialization file we provided you with to normalize columns width, etc., and run the script with the **SQLite** `read <filename>` command, where `<filename>` in the example above would be `Atan_Luring_LW1.sql`.

```
sqlite3 -init sqliterc mondial.db < Atan_Luring_LW1.sql
```

On successful completion, the file called (in this example) `Atan_Luring_LW1.log` would have the following content (modulo formatting):

```
select * from country limit 4;
Name        Code        Capital     Province    Area        Population
----------  ----------  ----------  ----------  ----------  ----------
Albania     AL          Tirana      Albania     28750       2800138
Greece      GR          Athina      Attikis     131940      10816286
Macedonia   MK          Skopje      Macedonia   25333       2059794
Serbia      SRB         Beograd     Serbia      77474       7120666
```

In the case of the **RA** software:

```
java -jar ra.jar mondial.properties -i Atan_Luring_LW1.ra -o Atan_Luring_LW1.log
```

runs the RA executable (ra.jar), taking as input the query in the file `Atan_Luring_LW1.ra` and writing the output to `Atan_Luring_LW1.log`. The homepage of **RA** is:

```
https://users.cs.duke.edu/~junyang/ra2/
```

[1] https://users.cs.duke.edu/~junyang/ra2/

Mondial

You can find documentation about Mondial in its website[2].

SQLite

For **SQLite**, the documentation on the website is comprehensive, but if you prefer learning from books, you have free access (from an UoM IP address) to this one:

> The Definitive Guide to SQLite
> Grant Allen, Mike Owens
> ISBN: 978-1-4302-3225-4
> Apress, 2010
> http://link.springer.com/content/pdf/10.1007%2F978-1-4302-3226-1.pdf

[2] `https://www.dbis.informatik.uni-goettingen.de/Mondial/`

Lab Work QO

This lab involves an experimental investigation into the **SQLite** query optimizer. Before embarking on the experiments it will be useful to refer to the **SQLite** documentation:

https://www.sqlite.org/docs.html

and in particular, this section:

https://www.sqlite.org/optoverview.html

Consider, also, studying this section in detail:

https://www.sqlite.org/queryplanner-ng.html

Task 1: Write **three pairs** of semantically-equivalent SQL queries against the Mondial database such that:

- in each query pair, either the two queries are syntactically distinct (but semantically equivalent, i.e., they return the same result set) and one query in the pair is optimizable by **SQLite** whereas the other is not;
- or else the two queries are syntactically identical but are run under different conditions (e.g., once without access to an index, and again with access to an index)

so that, as a result, we would expect the response times in evaluating the queries in the same pair to be different. Essentially, we want to know by how much, and why.

You must:

- produce the query pair,
- explain which optimization/evaluation strategy you have in mind for each query in the pair,
- record the execution times for each query in each pair,
- discuss any execution time difference.

You must use **SQLite** to evaluate your expressions and obtain results for your submission.

Task 2: Summarize your investigation with a plot, accompanied by interpretation and comment, that evaluates the benefits/shortcomings of using the **SQLite** optimizer and evaluator.

Software/Data

A folder is provided in Blackboard that gives you access to relevant data and scripts.

Some Comments

We expect you to study the documentation, i.e., to find out about facts, notions, ideas, techniques, policies and heuristics yourself. We want to gauge how much you have learned about query optimization and execution. Some topics (like the use of indexes) haven't been covered in detail in the course, but do some research.

By all means, ask questions in the lab session and in the discussion board, but we are asking you to think and reflect and ask yourself questions, propose some answers, and verify whether your proposed answers are correct or not.

Note, to start with, that the goal is not for you to try and create complex (in the sense of challenging, or even tricky) queries *per se*, but rather to study the **SQLite** optimizer and identify cases where it can make a difference in potentially speeding up a query and cases where it cannot. If you read them from this course unit's viewpoint, the documentation on the **SQLite** optimizer (sometimes implicitly) describes conditions in which the optimizer can make an impact and conditions in which it cannot make an impact.

To get you started, study the conditions under which the optimizer may or may not be able to use an index. Start by considering whether there is a substantive difference in the amount of data that would be processed if the index were used or not. Also, in the case of indexes, note that you may need to create the index yourself and run the query, then drop the index and run the query again. This is the special case in which the queries in the pair are the same but the context in which they were run (e.g., which indexes were available) is the difference.

For another example, consider how **SQLite** allows you to control whether or not its join ordering strategy is to be used. The optimizer has a join ordering algorithm. If you write a query that would benefit from reordering but the query explicitly uses `CROSS JOIN`, then the optimizer does not reorder. This should produce a difference, but it is not guaranteed to do so for all queries.

In short, the three query pairs are meant to be the result of your study of the effects of the **SQLite** query optimizer in the sense that you create one query in which the optimizer can make a difference and another (or another execution context) in which it is unable to.

Note that one shouldn't normally be very ambitious in terms of the impact of optimization strategies. So, an impact of an order of magnitude or more (i.e., ten, a hundred, etc., times faster) is rare, though it can happen. More commonly, an impact of 10% or more is noteworthy. A measurable but small impact (say, of between 5% or 10%) should perhaps be taken as encouragement for you to keep exploring, but, if you get stuck, you can use that. An impact of less than 5% veers into noise territory, i.e., it could have been caused by extraneous load in the machine and is, therefore, less useful in the context of this lab exercise. You can usefully run queries several times to obtain an understanding of the level of variability. You may also find more significant effects with larger databases, and we describe how to create larger databases that replicate the Mondial database in the Blackboard download for this lab.

What is most important is that you strive to show that you have understood the potential importance of a query optimizer, not so much that you can demonstrate the magnitude of the impact with **SQLite** on queries over the Mondial data. Therefore, consider (in your plot and your analysis) making explicit the percentage differences in addition to the absolute times and don't necessarily expect very large percentage differences.

Note, however, that if you were to produce three pairs of queries that are simply instances of the same hypotheses (e.g., "'The availability of indexes allows the optimizer to generate more efficient plans.") you would lose out on some marks. Contrast the case above with one in which you explore different optimization strategies that are based on indexing (e.g., one might be the case of complex predicates that prevent the optimizer from taking advantage of indexes, another the case where the availability of an index allows the optimizer to select a more efficient physical operator, etc.). In this example, the hypotheses are different, and the corresponding queries aim to trigger different behaviours in the optimizer, and hence merit more marks.

Your submission will be a report containing the pairs, the explanation of the optimizations you have in mind, and the plots with interpretation and comment.

Task 1: Query Analysis:

The report should be structured with a section for each pair of queries that includes:

- The pair of queries. The queries should be in a form that can easily be copied and pasted into a script, in case we want to run it.
- An explanation that describes the features of the SQLite optimizer and evaluator that are being investigated, and the impact of these on the times observed.
- Evidence that has been collected that the pair of queries provide different execution times.

Task 2: Reflection:

- At least one plot that provides information on the approaches to evaluation pursued for Task 1.
- A discussion of the plot, the information that was used in the experiment, what can and can't be observed from this information, and what other information might be useful.

Thus, the discussion in Task 1 relates to the specific queries, and the discussion in Task 2 seeks to draw more general lessons from your overall experience, in the light of the information in the plot.

The marks are allocated 75% to Task 1 and 25% to Task 2. **The total length of the report should be no longer than 2000 words.**

On Plotting, Interpreting and Commenting on Experimental Results

You are doing a study in which you measure the response time of queries. So, what goes in the Y axis? And you're plotting the response time for various queries, which come in pairs. So, what goes in the X axis? In other words, think about this as an opportunity for you to exercise/acquire basic transferable skills in doing empirical explorations of system performance that might be useful in non-database settings.

To throw further light on what you're expected to do when you're asked (above) to *explain the optimization you have in mind*, the starting point for you is to think of these tasks as being grounded on the notion of hypothesis testing.

So, you state what you expect to happen given some initial conditions and then you take measurements to see whether the actual measurements obtained provide evidence one way or the other.

To produce a plot, you can use a spreadsheet program, or else a program like `gnuplot`[3]. For this course unit, the plots can take the form of a bar chart with each query corresponding to a bar (paired with another, in some cases). Always label the axes, and, if pertinent, provide the unit of measurement (in timings, this is typically seconds).

Beware extremes of range in the Y-axis, otherwise you could find yourself producing visually misleading plots. This means that your analyses may require you to produce additional zoomed-in plots that adjust the X and Y axes to a narrower, and/or shorter, range in one of the axis, or both, in order to focus on and highlight interesting similarities or interesting differences, divergences, or contrasts.

In your comments, make a point of thinking about one to three most important findings that result from the experiment. For example:

> The SQLite optimizer consistently improved performance by at least one order of magnitude.

Whenever possible aim to highlight similarities and contrasts. Whenever possible aim to present absolute values but also percentages. For example, not only:

> The optimized query took 3ms while the non-optimized query took 3.5ms.

but also

> The non-optimized query, therefore, took approximately 17% longer than the optimized one.

Your comments should avoid simply reading the values off the graphs. For example, don't simply write

> In the first pair, the non-optimized query took 3.5ms.

the reader can do that without your help (unless you plot is poorly presented, which is another, serious, matter).

Instead, aim to link the results with what you are learning in the course. For example,

> The use of both join reordering and indexing in the optimized query results seems to produce less significant reductions in response time than one might have expected, one reason for which might be the peculiar distribution of the data, something that would need to be investigated further.

When reporting times, you should provide information about the setup of the experiment, including the database size, the software version, and information about the environment used. For example, here's a pretty complete specification of a (hypothetical) underlying hardware/software environment:

[3] http://www.gnuplot.info/

These measurements were taken in a MacBook Air, 1.8 GHz Intel Core i7 with 4GB 1333 MHz DDR3 with 251 GB Flash Storage running OS X Yosemite V 10.10.5 and SQLite version 3.8.11.1 2015-07-29 20:00:57.

You may not have as much information, but provide as much as you can on the CPU, the primary memory, the secondary memory, the operating system and the DBMS you're using.

Sometimes, a lot of time may be spent in printing results. For exercises in which the result is not of special interest, only the timings are, you should switch off the echoing in your script.

Furthermore, because of hot-cold effects (i.e., full v. empty buffers, on-the-fly indexing, etc.), when taking measurements of query response time, it is good practice to (close and re-)open the database anew before running each query, and then run each query four times at least, discarding the first and averaging the last three to obtain the value you plot.

If running on a public machine, to check whether there are other users logged into a machine, and thus potentially interfering with measured times, log into it and issue the `who` command on a shell: it lists the login names of every used logged into the machine.

When explaining how results have been obtained, it may be useful to use the SQLite explain query plan command[4]. The

```
        .eqp on
```

command switches on explanation, and will provide information on how a query is to be evaluated. For example:

```
sqlite> .eqp on
sqlite> select * from country limit 4;
QUERY PLAN
`--SCAN TABLE country
```

[4] https://www.sqlite.org/eqp.html

Lab Work SP

In relational databases, the content of every table must conform to an explicitly declared schema. In semi-structured databases, including RDF stores, however, this is not the case.

The overall, high-level goal of this lab work is for you to gain an insight into some of the consequences ensuing from the lack of schema constraints over RDF stores. In particular, if one is asked to write a set of SPARQL queries to achieve a certain goal, some questions arise, such as:

- How does one go about understanding what data is there, and what can we ask of it?
- How much time is spent finding out?
- Does this make the task of writing queries harder?
- Is it more likely that you will spend longer debugging the queries before they work?

So, there needs to be an exploratory stage in which you, the query writer, must familiarize yourself with the opportunities (i.e., the subjects, predicates and objects) that are available in one or more RDF datasets. You can do this using SPARQL itself, of course, but is this effective? Is it efficient?

In this lab work, we will fix a starting point (viz., DBpedia (https://www.dbpedia.org), i.e., the linked data version of Wikipedia) and a target information content for you, and you will aim to create the queries that derive that content from one SPARQL endpoint.

Note that some of the desired information content will not be available (you will need to convince yourself of that and then explain why in your report). Note too that some of the desired information content may well be there but you cannot find (perhaps in the time available) a way of reaching it (e.g., some predicate may be missing and therefore linking/navigating to the desired content proves impossible or exceedingly hard).

In terms of information content, your goal is to create a set of SPARQL queries over DBpedia that, for a country in the relational version of the Mondial database, generates the same information (excluding the geographical aspects such as lakes, mountains, seas, rivers, etc., but including, if possible continent, country, city, province, organization, ethnic group, language and religion) as is available in Mondial.

In other words, you will:

1. study again the relational version of Mondial,
2. focus on the non-geographical information that you can retrieve (basically using selection-projection-join-aggregation), as shown in the `mondial-abh.pdf` referential dependency diagram, then
3. set yourself the target of obtaining that information from DBpedia using SPARQL queries.

Note that we will not be strict about minor variations (e.g., the names given to countries, the populations of a city, and similar cases), minor inconsistencies (e.g., the province to which a city belongs, and similar cases), and minor examples of incompleteness (e.g, some missing cities or missing properties, and similar cases).

Note also that there is a version of Mondial in RDF, but you will *not* use it. You will try and create, from DBpedia and using SPARQL, the information content of the relational Mondial restricted to what you can obtain by querying country, City, etc, and ignoring geographical information (i.e., information about geographical features and concepts such as lakes, rivers and islands).

However, creating the set of queries is not your main task. Moreover, the task for which most marks are available is *not* to get the set of SPARQL queries absolutely right.

Your main task is to write a report about your experience of writing the set of SPARQL queries. In other words, the main goal, really, is for you to record, reflect and report on your experience of writing the SPARQL queries over DBpedia.

Here, what we mean to learn is, given the need for exploration of possibilities and opportunities (which, as we said, is one consequence of the schema-less nature of SPARQL/RDF) *how effective and how efficient you managed to be*

By

how effective

we mean an answer to the question:

To what extent did you succeed in writing a set of SPARQL queries that gathers from DBpedia the same information that is available through relational Mondial?

Note that we are not expecting you to be 100% effective: you may not be able to retrieve all the information, your set of SPARQL queries may be incomplete.

Also note that, here, you don't need to worry about single tuples, single values, single pieces of information. We're only interested in questions such as: *Can we find the provinces of a country using SPARQL against DBpedia?* Note that we don't necessarily mean *all* the provinces of *all* the countries.

By

how efficient

we mean an answer to the following questions:

- *How much time did it take you to explore of the information content of the DBpedia using SPARQL queries that try and gauge what is available in that resource?*
- *How much time did it take you to discover how to write the set of SPARQL queries you have succeeded in writing?*

Again, note that we are not asking you to be extremely efficient: it may take you quite long to even begin to write the first SPARQL query (most probably, one that tells you which are the countries in DBpedia).

You don't lose marks if it takes you very long. You gain marks for explaining what the problem was that prevented you, be it a problem with DBpedia, be it a problem of complexity (e.g., you ran out of time before the submission deadline), be it a problem of inadequacy of SPARQL, be it the lack of a schema, or yet something else.

Also note that, here, you don't need to worry about very detailed timekeeping. Just keep a timesheet of the amount of hours and minutes (down to 5 minutes accuracy) that you spend in this lab.

For example, you may have several rows saying something (slightly more specific perhaps than)

1) Exploring which predicates may be relevant to Country: Monday 13:00-13:30
2) ...
3) ...
4) Exploring which predicates may be relevant to City: Monday 18:00-19:30

and then do some aggregation and commenting, e.g., something like

TASK	TIME SPENT	COMMENTS
...		
Exploring which predicates may be relevant	2 hours	(a) DBpedia was down often, (b) It took me long to find a path from country to province
...		

Don't take the hypothetical numbers above as any kind of hint: they're invented numbers for the sake of an example. Such tasks may take you much less time or much more time. Finding out the (rough, approximate) time is one of the goals in this lab work.

Task 1: Write a set of SPARQL queries for the task described above, compiling a timesheet as you do, and writing comments on the efficiency and effectiveness of your work on the task.

The queries will be typed, tested and executed in the landing page for the DBpedia SPARQL endpoint:

http://dbpedia.org/sparql

You will find it useful to also consult the DBpedia Ontology directly:

http://mappings.dbpedia.org/server/ontology/classes/

We expect that you will use SPARQL queries for two related, but distinct, purposes: one purpose is to explore how the information content of Mondial is (or is not) made available in DBpedia (call these *exploratory queries*), the other is to actually retrieve data from DBpedia that corresponds to data you might have retrieved from Mondial using SQL or XQuery (call these *retrieval queries*). Note that, in the absence of a schema, one needs to use the former kind of SPARQL query to obtain the information needed to write the latter kind of SPARQL query.

Task 2: Write a report that includes the set of queries produced in Task 1, and, using the timesheet and comments compiled during Task 1, assesses the impact that the schema-less nature of SPARQL/RDF had on the *effectiveness* and *efficiency* of your work on Task 1.

Some Comments and Tips

Using exploratory queries

By *exploratory query* is meant one that essentially is trying to explore (a sample of) what subjects, predicates and objects there are. One query that might get you started is the following:

```
SELECT DISTINCT ?C
WHERE {?C a dbo:Country}
```

Try it. You would expect this to be a list of countries, right? But while indeed resources representing countries are returned, e.g.

http://dbpedia.org/page/Brazil

we also get information that is not about a country *per se*, e.g.

http://dbpedia.org/page/Captaincies_of_Brazil

(These are just illustrations! You don't need to stop and understand the resources the links above point you to. It is just so that you begin to get in thinking mode!)

We expect that a significant part of your time (but prove us wrong!) will be spent exploring the content of DBpedia this way. You are asked to capture and report on the process of exploring DBpedia.

Being aware of the namespace prefixes you can use

In the landing page, the *Tables* drop down menu links to the predefined Namespace Prefixes; you should explore this.

Using 'Results Format' to help exploration

Remember that if you use HTML as 'Results Format', you are given links that you can click on and explore more. For example, the query above returns a table with HTML links. If you click on `http://dbpedia.org/ontology/Place`

you will be taken to a description of the concept `Place` in the DBpedia Ontology. This may tell you more about the data and how to use it to achieve your goal. There are other options than simple HTML. You should explore your options.

Using `LIMIT`

Be careful to use `LIMIT` otherwise you'll get a flood of results. On the other hand, you may need to increase `LIMIT` or remove it altogether to find what you're looking for.

About the timesheet and comments

The timesheet and comments (see above) are important but do not worry if the times are not extremely accurate, or if you forget to make a note of one among ten things you did, etc. It's crucial that your report reflects your own thoughts on whether the schema-less nature of SPARQL was (or was not) an impediment to your being effective (i.e., getting the queries right) or efficient (i.e., getting (some of) the queries in not overly long times).

About the open-ended nature of this lab work

Finally, keep in mind that the goal of this lab is not to get the queries right at all costs, though you should make a serious attempt at populating the relevant Mondial tables; the goal is for you to record, reflect and report on your experience of writing SPARQL queries over DBpedia that aim to meet certain data requirements (viz., recreating the non-geographical information content about countries, cities, etc, that you can obtain from the relational version of Mondial).

Marking

- We suggest that you write your report in a way that includes a structure in which you:
    - Repeat for each Mondial concept covered (e.g., Country, City, …) the following information:
        - An outline of the item to be populated from Mondial.
        - The exploratory queries with timesheet information and comments.
        - The retrieval queries with timesheet information and comments.
        - Any additional comments on effectiveness and efficiency of the results.
- The marks will be awarded for the quality of the report on the effectiveness and efficiency of the task as described in the preamble above, including:
    - The scope and suitability of the exploratory queries and the depth of the discussion on efficiency and effectiveness.
    - The scope and suitability of the retrieval queries and the depth of the discussion on efficiency and effectiveness.
    - The clarity of the timesheet and the quality of the associated discussion.

The total length of the report should be no longer than 3000 words.