CS392 Database System Concept – Project

Final Report Due: May 30, 2014

Introduction

This is a group-based mini research project. You are given three type of data sources. Each group picks one data source of your choice. By **Monday, March 31**, each group must email Jack (jacksunwei@gmail.com) your choice of data source. If a data source is chosen by too many groups, it will be allocated on a first-come, first-serve basis.

You task is as follows:

1. Study and understand the chosen data source, and design some useful queries.
2. Design a relational database (a set of schemas) for the data, so that you can load the data into database and let database to support the queries you designed and the queries listed in the requirements.
3. Verify the design of your models. You should optimize your design, so that the database can return results of the queries more quickly.
4. Implement a simple graphical user interface to perform your queries, so that I can check the performance of your design.
5. Write a report to describe your design and implementation and hand out all source code (MySQL script as well as GUI code).

Specifications

1. Data

Each given data source is accompanied by a description and some example queries (known as requirements).

Description will be the basic information about the data, such as the format of file, the structure of the data, the meaning of the fields in data, etc.

Requirements are a number of queries written in English. These must be supported by your database design efficiently.

2. Model

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model, which uses a table-based format. (See Wikipedia)

You need to design a model for the given data, and map the model to schemes of a real relational database (MySQL).

3. Optimization

First rough version usually cannot satisfy the demands most. In order to make your model fast or efficient for your proposed queries. You need to revise your model design and adjust parameters of database to make your database faster and faster. The potential optimization can include but not limited to refining table design, building index, distributing nodes, etc. However, no matter what kind of optimization you use, you should give the persuasive reason that you really need to do it and it indeed take effects. In other words, you need to design solid experiments to demonstrate your design.

4. Experiment Design

Experiments are designed to demonstrate your ideas. An effective experiment should at least contain the following parts:

- Hardware Specifications
- Dataset
- Test Queries
- Initialization Scripts
- Experiment Procedure
- Result Analysis

For each of your optimization, you should design an experiment to demonstrate the advantages and disadvantages and whether you will take that optimization as part of your design.

Say, an index might improve speed of query. However, it also takes disk space to store them. As a result, if one column is not like to be filtered, there is no need to build index on it.

In case you still do not know how to design or set up experiment, you can refer to the first few chapters of *High Performance MySQL*.

5. Graphical User Interface

This is essential for us to check the performance of your design. There is no need to make it very fancy, but it should be able to let others perform your queries with different parameters.

Presentation

Depending on the schedule of this course, you may be required to present your ideas during the last class. The presentation should include your basic ideas and expected experiments. No concrete data is needed.

Deliverables

The final deliverables should include the following items:

- A well-written report to describe your ideas, design, experiments, conclusion, etc.
- Initializing scripts of database system (MySQL).

- Source code of GUI.

Scoring Criteria

Your score will consist of three parts.

The first one is from the designed queries, which holds **45%**. They should be meaningful and expressive.

The second one is from the design and optimization of your database model. It should contain your sufficient concern and well-designed experiments to demonstrate your design. This part holds another **45%**.

The last one is from the graphical user interface. It can be simple, but strong to test your design with different parameters. This parts holds **10%**.

Data I – Probase (link, online demo)

Description

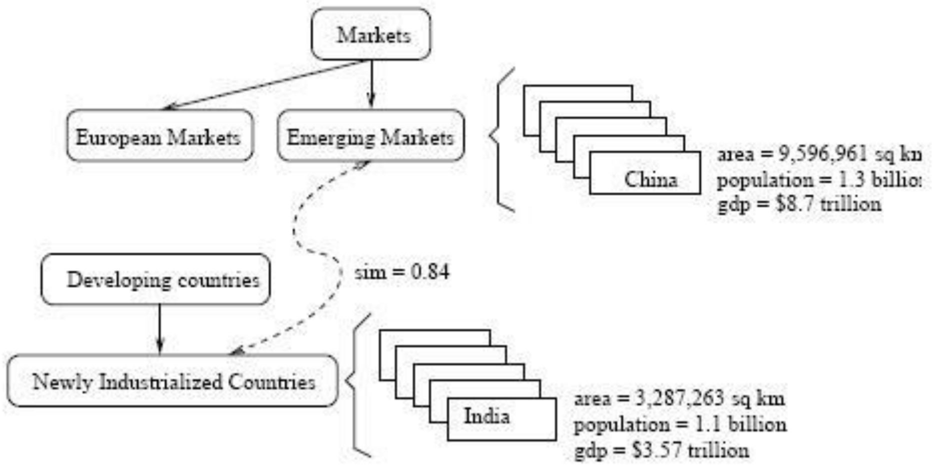

Figure 1: A snippet of Probase's core taxonomy

Figure 1 shows what is inside Probase. The knowledgebase consists of concepts (e.g. emerging markets), instances (e.g., China), attributes and values (e.g., China's population is 1.3 billion), and relationships (e.g., emerging markets, as a concept, is closely related to newly industrialized countries), all of which are automatically derived in an unsupervised manner. In this project, we focus on the data that contains Is-A relations.

Data Format: Is-A Relation Data

1. File: Isa_Core.txt
2. Format: <Concept>\t<entity>\t<frequency>\t<popularity>\t<ConceptFrequency>\t<ConceptSize>\t<ConceptVagueness>\t<Zipf_Slope>\t<Zipf_Pearson_Coefficient>\t<EntityFrequency>\t<EntitySize>
3. Columns:
   - **Concept** is the Hypernym in Hearst pattern or isa pattern. For example, “country” will be extracted as a concept from sentence “Countries such as China, ….”.
   - **Entity** is the hyponym in Hearst pattern or isa pattern . For example, “China” will be extracted as an entity from sentence “Countries such as China, ….”.
   - **Frequency** is the occurrence of the pairs in the sources documents.
   - **Popularity** is the number of different domains where we found the given pair.
   - **Concept frequency** is the occurrence of the concept in the sources documents.
   - **Concept Size** is the number of the unique entities in the concept.
   - **ConceptVaguenessScore** denotes how vague the maning of this concept is. Smaller score means Vaguer.

- **Entity frequency** is the occurrence of the entity in the sources documents.
- **Entity Size** is the number of the unique concept containing the entity.
- **Zipf's Law** states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. For a given concept in Probase, let:
  - *N* be the number of entities of the given concept;
  - *k* be the rank of an entity in the concept;
  - *s* be the value of the exponent characterizing the distribution.

    Zipf's law then predicts that the frequency of the entity with rank *k*, *f*(*k*;*s*,*N*), is:
- **ZipfSlope:** s in above Zipf formular. Zipf's law applies when s = 1.
- **ZipfPearsonCoefficient** PearsonCoefficient between log(f) and log(k). When PearsonCoefficient close to 1 or -1, log(f) has a linear relation with log(k).

Requirements:

1. Given an entity, analyze the distribution of its concept.
2. Given a concept, analyze the distribution of its entity.
3. Given two entities, analyze the distribution of their shared concepts.
4. Given two concepts, analyze the distribution of their shared entities.

Data II - Freebase

Description:

Freebase is a large database aiming at describing the entities in the real world, including the type of an entity, the attribute of an entity, and relations between entities. In Freebase, both attribute and relation is called property, and do not distinguish between them.

The data in Freebase is organized as a triple form, such as is:

<obj1, property, obj2>.

Here both obj1 and obj2 can be entity, type, or property. Further more, obj2 can be number, label or enumeration.

For example, Freebase can tell us the fact that "the capital of China is Beijing", because Freebase contains the property "capital" and two entities "China" and "Beijing".

The corresponding triple (one row in Freebase data) is as follows.

Example 1:

<http://rdf.freebase.com/ns/m.0d05w3> <http://rdf.freebase.com/ns/location.location.capital> <http://rdf.freebase.com/ns/m.01914>

(NOTICE: we will use "FB/" to replace the "http://rdf.freebase.com/ns/" prefix in the following examples, because it's too long.)

The first item (obj1) is an entity representing the ID of "China"; the second item (relation) is the property ID of "capital", and the last item (obj2) is also an entity, the ID of "Beijing". As you may know from the example above, entity / type / property in the dataset are represented by their unique ID.

Freebase has more 10K properties, such as "capital". Besides these real-world properties, freebase has several important properties to construct the data framework. Here we introduce some of them:

1. The name of an object: ***type.object.name***

   Freebase uses this property to connect an ID with its name.

   Example 2: ***<FB/m.0d05w3> <FB/type.object.name> <"China"@en>***

   Here @en indicates the language of the name (similarly, *@fr* is the indicator of French name, and so forth.) We only consider English names.

   Example 3:

   <FB/location.location.capital> <FB/type.object.name> <"Capital"@en>

   A property ID also has names. So does a type.

2. The type of an object: ***type.object.type***

   Every object in Freebase has its types. When a triple describes this relation, **obj2 is always a type**.

Example 3: *<FB/m.0d05w3> <FB/type.object.type> <FB/location.country>*

obj1 is "China", obj2 is the ID of type "country"

Example 4: *<FB/m.0d05w3> <FB/type.object.type> <FB/royalty.kingdom>*

One object can have several types.

Example 5: *<FB/location.location.capital> <FB/type.object.type> <FB/type.property>*

A specific property ("capital") is a property. Note that obj2 is a **type**, which has the name "property".

Example 6: *<FB/location.country> <FB/type.object.type> <FB/type.type>*

A specific type ("country") is a type. Note that obj2 is also a **type**, which has the name "type". Example 5 & 6 are a little twisted, but I hope you can use this relation to understand what a specific object ID represents.

3. Instantiation of a type: ***type.type.instance***

   Freebase uses this relation to get all entities of one type.

   Example 7: *<FB/location.country> <FB/type.type.instance> <FB/m.0d05w3>*

   Easy to understand, isn't it?

4. Type constraint of one property: ***type.property.schema***

   type.property.expected_type

   Every property actually describes the relation between an entity in one type, and the other entity in another type. For example, "capital" describes the capital_of relationship between a country and a city. Which means, for every <obj1, relation, obj2> triple where the relation is "location.cou-ntry.capital", obj1 must have type "country" and obj2 must have type "city". In order to represent this restriction, Freebase uses these two properties to connect the types with one property. The type of obj1 is called "schema", and the type of obj2 is called "expected_type".

   Example 8:

   <FB/location.country.capital> <FB/type.property.schema> <FB/location.country>

   The obj1 in the property "capital" must be a country.

   Example 9:

   <FB/location.country.capital> <FB/type.property.expected_type> <FB/location.citytown>

   The obj2 in the property "capital" must be a city/town.

   For more information about the structure of Freebase, please refer to the following websites:

   https://developers.google.com/freebase/v1/rdf-overview

   https://developers.google.com/freebase/data#freebase-rdf-dumps

Requirements:

1. Search a name, return all **entities** that match the name, better to rank the entities, like Wikipedia. For example, when we search Einstein, the famous physicist Albert Einstein may rank first.
2. Given an entity ID, return all types it belongs to.
3. Given an entity ID, return all properties whose schema/expected_type is one type of the entity.
4. Given an entity ID, return all objects that are co-occurred with this entity in one triple.

Data III – Yago

Description

YAGO is a huge semantic knowledgebase, it contains more than 10 million entities and more than 120 million facts about these entities.

Each piece of knowledge in YAGO is represented by a triple form. For example, one relation between entity1 and entity2 is represented as <entity1, relation, entity2>. This kind of triple is called a fact, which builds the bridge between two different entities. Besides, every entity belongs to one ore more types defined in YAGO, which can also be represented as triples. Further more, one fact can be associated with temporal and geospatial information, if possible. YAGO2 (the latest version of YAGO) contains this kind of information by connecting a fact (using fact ID) with a specific time or a location.

For the detail information about YAGO and YAGO2, please refer to the following papers:

[1] F. Suchanek et al., YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. WWW, 2007.

[2] J. Hoffart et al., YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. Artificial Intelligence, 2012.

You can freely download the YAGO2 data here:
http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html

Requirements:

1. Given a name, return all the entities that match the name.
2. Given a fact ID, return the entity1, relation, entity2 in the fact.
3. Given a type, return all the relations whose domain or range matches this type.
4. Given an entity, return all the relations, and the other entity/fact that is co-occurred in one triple.
5. (optional) Given an entity, return all the types it belongs to.
6. (optional) Given a fact ID, return the geospatial or time information that is connected the fact, as well as the connecting relation.