BUILDING DATABASE FOR NELL DATABASE2015

by

5120309700 Qian Kun

5120309706 Lou Jiong

5120309707 Han Jianhua

Submitted the final project report of
Database Technology course

at

Shanghai Jiaotong University
Minhang, Shanghai
November 2015

Table of Contents

Chapter 1

Introduction of database

1.1 NELL

NELL (Never-Ending Language Learner) is a computer system that learns over time to read the web. So far it has accumulated over 50 million candidate beliefs by reading the web, and it is considering these at different levels of confidence.

1.2 Meaning of every column

For each column of file, it represents:

Entity:The Entity part of the (Entity, Relation, Value) triple. Note that this will be the name of a concept and is not the literal string of characters seen by NELL from some text source, nor does it indicate the category membership of that concept.

Relation: The Relation part of the (Entity, Relation, Value) triple. In the case of a category instance, this will be "generalizations". In the case of a relation instance, this will be the name of the relation.

Value:The Value part of the (Entity, Relation, Value) triple. In the case of a category instance, this will be the name of the category. In the case of a relation instance, this will be another concept (like Entity).

Iteration of Promotion, Probability, Source, Entity literalStrings, Value literalStrings, Best Entity literalString, Best Value literalString, Category for Entity, Category for Value, and **Candidate Source**.

Chapter 2

Platform

For this task, we use python to preprocess our data, and phpmyadmin to store our data. We also use php to build our user interface.

1. Apache 2.2.21
2. php 5.3.10
3. Mysql 5.5.20
4. phpmyadmin

Chapter 3

Data prepossessing

Before we upload the data into the database, we use python to get us some new table from the initial table

1. To deal with the first query, we must preprocess the raw data to get the table(Name,Entity) we want. In that case, we first need to scan the whole table to fetch the Entity and negelect the urls. After the first step, we cut the last part of Entity as Name. Finally, each pair of Name and Entity should be stored in our new table.

2. For the second query and third query, we want to get the table(Entity,Category). And according to the requirement, we want to have another element Probability for filtering and ordering. So in this part, we scan the whole Nell table to fetch Entity, Category and Probability where the relation of these entities are generalizations. Then we check each Entity and Value, so that we can find the columns containing the whole categories of them. The last step of creating this table is to split the string of raw category and store them with corresponding Entity and Probability.

3. The last table is for query 4. We want to build the table(Entity, Related_Entity). Again, we scan the whole raw data and fetch each pair of Entity and Related_Entity where the relation is not generalizations. Then, we also nelgelect the pairs that containing urls. Finally, storing all the pairs so that we get table(Entity, Related_Entity).

Chapter 4

Schema and E-R model

To solve the basic query problem, our schema likes this:

Entity_schema = (Name, Entity)

Belong_schema = (Entity,Category,Probability)

Relation_schema = (Entity, Related_Entity)

And our E-R model:

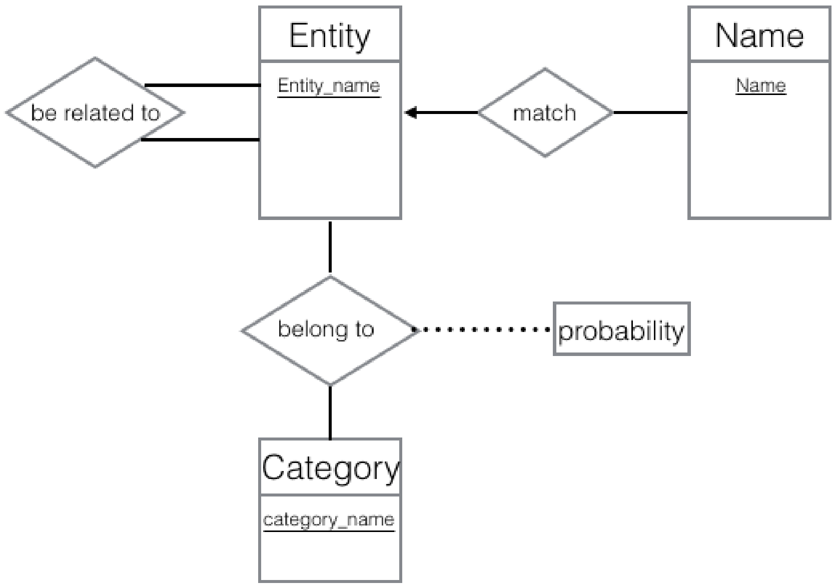


Figure 4.1: E-R model

Chapter 5

Experiment

5.1 Basic query

1. Given a name,return all the entities that match the name.

sql:

```
SELECT Entity
FROM ‘entity‘
WHERE Name = ”;
```

2. Given a category,return the top 50 typical entities belongs to this category.

sql:

```
SELECT Entity
FROM ‘entity_category‘
WHERE Category = ” order by Probability desc;
```

3. Given an entity, return all categories it belongs to.

sql:

```
SELECT Category
FROM ‘entity_category‘
WHERE Entity = ”;
```

4. Given an entity, return all entities that are co-occurred with this entity in one triple.

We use two different sql to solve this problem.

sql1:

```
SELECT *
FROM entity_entity
WHERE Entity = ”
```

```
union
SELECT *
FROM entity_entity
WHERE Related_Entity = ”;
```

```
sql2:
SELECT *
FROM entity_entity
WHERE Entity = ” or Related_Entity = ”;
```

5.2 Additional query

Besides all of this basic queries, we find two more interesting queries that can be done with our schema.

1. Given a name, return all categories it belongs to.

```
sql:
SELECT Category
FROM
(SELECT *
FROM ‘entity‘
WHERE Name = ”) NATURAL JOIN ‘entity_category‘;
```

2. Given a category, return the pair of related entities belonging to it.

```
sql:
SELECT entity_entity.Entity, Related_Entity
FROM entity_entity, entity_category as t1, entity_category as t2
WHERE entity_entity.Entity=t1.Entity and entity_entity.Related_Entity = t2.Entity
and t1.Category = ”;
```

5.3 Optimization

5.3.1 Remove duplication

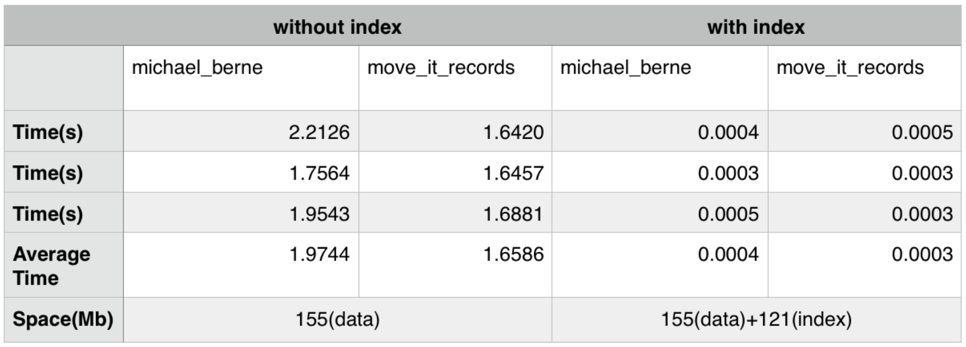
We use mysql to remove duplication, for example, we want to remove the duplicated tuple from table1:entity, what we do just like this:

create table test1_distinct as (select distinct * from ‘table1‘);

So we can get the test1_distinct. In result, the query time will be shorter, and the storage it need will get much smaller.

5.3.2 Build index

Building index must be a good way to reduce query time.We will just use table1:entity as an example to show the change it makes.



	without index		with index	
	michael_berne	move_it_records	michael_berne	move_it_records
Time(s)	2.2126	1.6420	0.0004	0.0005
Time(s)	1.7564	1.6457	0.0003	0.0003
Time(s)	1.9543	1.6881	0.0005	0.0003
Average Time	1.9744	1.6586	0.0004	0.0003
Space(Mb)	155(data)		155(data)+121(index)	

Figure 5.1: result comparation

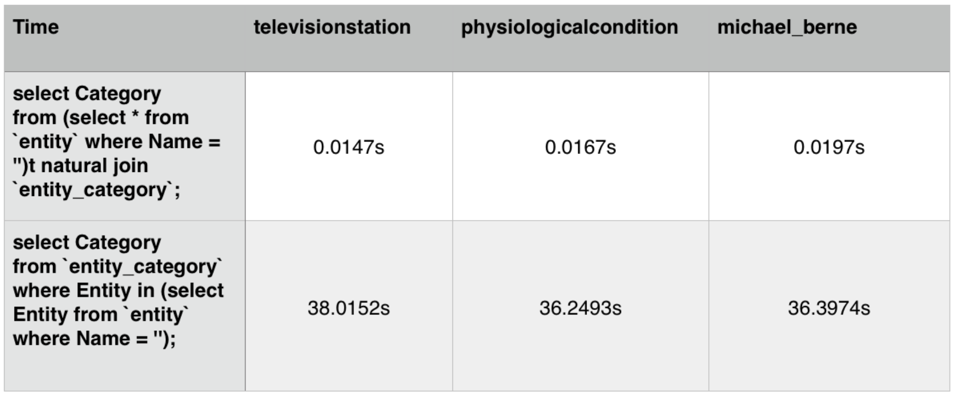
So we can see from this table, the time will be highly reduced if we build an index on name. But on the contrast, it will need more space to store this index and we need about 27.8s to build this index, so its a trade off between time and space.

We also did some further optimization when building the index. In query2, what we build is a composite index. A composite index is an index on two or more columns, for example we build the indexcategory probabilityfor query2. Composite indexes are especially useful in two different circumstances. First, you can use a composite index to cover a query. Secondly, you can use a composite index to help match the search criteria of specific queries. Contrast to the normal index we used before, the composite index takes less space on the disk. In query2 we ordered the data we found through the probability and search data through the category. In this case the composite index will give a better performance on both time and space.

5.3.3 Sql Optimation

In this part we use contrast experiment to show different sql statements influence on the performance of query. When doing optimization, we generally think from two aspects. One is which part is long-running queries and how can we isolate them, the other is how to identify the cause of long-running queries.

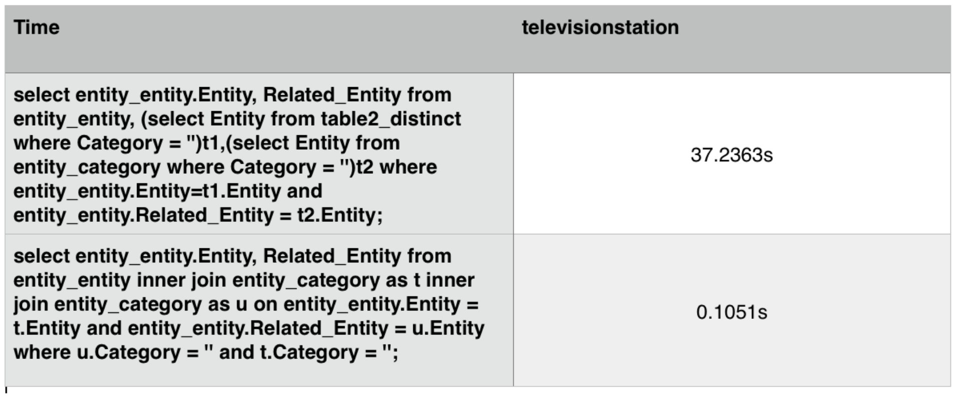
We use our query5 to show our thinking of sql optimization.In the designed experiment query5, we want the user to give us a name, and we will return all categories it belongs to. We use three different sql statement and test each sqls query performance. In our experiment, it is obviously that the SELECT FROM nested query is what we need to avoid from using because of its time consuming. Besides that, we also test the performance of using a subquery within the IN operator, which is really expensive because SQL query will evaluate the outer query first before proceed with the inner query. Our solution to that problem is using natural join, also we can use dummy table as another solution.



Time	televisionstation	physiologicalcondition	michael_berne
select Category from (select * from `entity` where Name = '')t natural join `entity_category`;	0.0147s	0.0167s	0.0197s
select Category from `entity_category` where Entity in (select Entity from `entity` where Name = '');	38.0152s	36.2493s	36.3974s

Figure 5.2: query5 results

And in query6, we want the user to give a category and return the pair of related entities belonging to it. In this case, we also give two different solution to run the query. In our first solution, we use the nested select form and since that we cant use the index we created before. Then we tried another way, we use nature join in order to use the index and the performance is a lot better than the former one.



Time	televisionstation
select entity_entity.Entity, Related_Entity from entity_entity, (select Entity from table2_distinct where Category = '')t1,(select Entity from entity_category where Category = '')t2 where entity_entity.Entity=t1.Entity and entity_entity.Related_Entity = t2.Entity;	37.2363s
select entity_entity.Entity, Related_Entity from entity_entity inner join entity_category as t inner join entity_category as u on entity_entity.Entity = t.Entity and entity_entity.Related_Entity = u.Entity where u.Category = '' and t.Category = '';	0.1051s

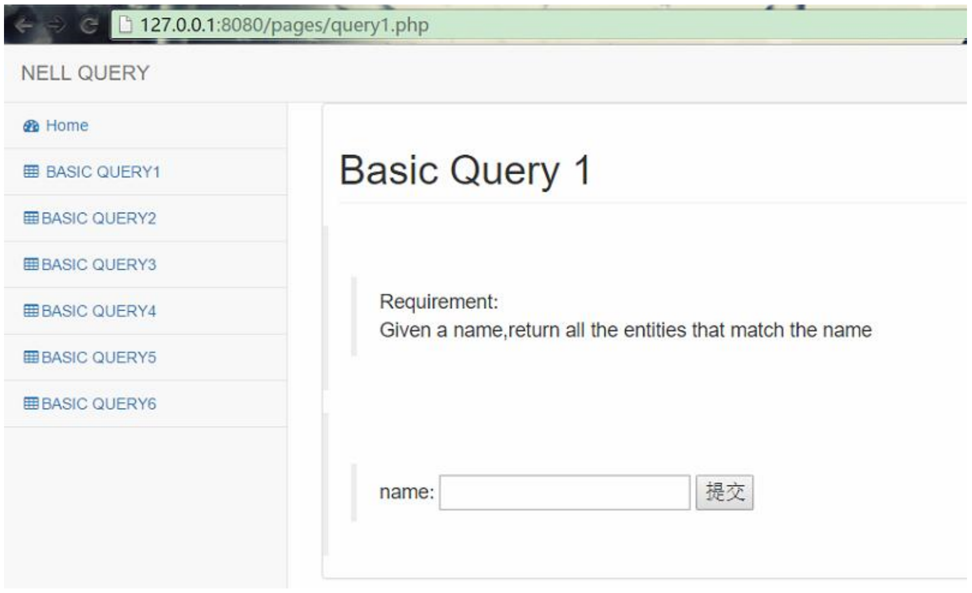
Figure 5.3: query6 results

Chapter 6

User interface

To make people, especially the people who dont know mysql, use our database easily, we use php to connect the database and build a simple GUI for our queries. We use the wampserver to build the LAMP environment for windows, including apache, mysql and php.

Here we use the query1 as an example to show how our user interface works.

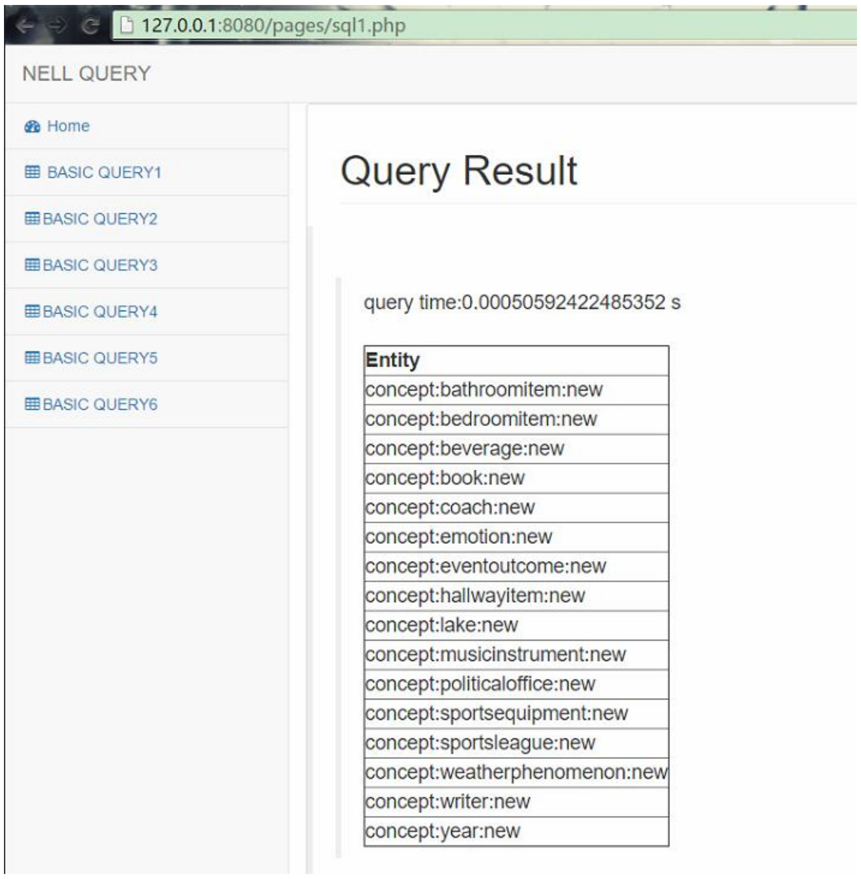


Basic Query 1

Requirement:
Given a name,return all the entities that match the name

Figure 6.1: Query1

As is mentioned before, we use php as our database interaction language, so we use mysql_query() to submit our sql statement to the database. The user can use our website to query for anything according to the requirement. In the first query webpage, the requirement ask for name and will return the matching entities, so users only need to type in the name they want to search and the result is shown in the next figure.



Query Result

query time:0.00050592422485352 s

Entity
concept:bathroomitem:new
concept:bedroomitem:new
concept:beverage:new
concept:book:new
concept:coach:new
concept:emotion:new
concept:eventoutcome:new
concept:hallwayitem:new
concept:lake:new
concept:musicinstrument:new
concept:politicaloffice:new
concept:sportsequipment:new
concept:sportsleague:new
concept:weatherphenomenon:new
concept:writer:new
concept:year:new

Figure 6.2: Query1 result

This figure shows the result of users query about word new. In this example, we return all the entities we get from the database, whats more we also return the time our program used when doing query.