SE305 COURSE PROJECT FINAL REPORT

Jeremy Liu, Qianyang Peng, Jingyu Cui

Contents

1 Model Design

Our models were designed with Workbench. The diagrams are much like ER diagrams but the relationships between entities are implicit. Besides the basic requirements, our design provides advanced functions including:

1. We store the label of entities and text of descriptions in **all languages**, not only in English.

2. For claims, we store the **qualifiers** in addition to the mainsnaks.

3. For all the snaks and qualifiers, we designed tables to store various types of **datavalues**.

Our design contains nine tables and satisfies BCNF. Moreover, we used *utf8mb4* dataset to support different language characters.

The ER Diagram can be found in Figure 1. Details of the design are included in the next section.

The programming tools we used:

1. Table design and create schema: MySQL Workbench

2. File parsing and data insert: Python3

3. User interface: Apache & PHP

2 Table Designs

We finished our table design according to our model design. Our final design contains 9 tables. Reference and aliases are not stored in our tables, while all other information in different languages are stored. The design of our tables is elaborated below:

2.1 entity

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    entity` (
  `serial_id` BIGINT(32) NOT NULL
     AUTO_INCREMENT,
  `entity_id` VARCHAR(32) NOT NULL,
  `entity_language` VARCHAR(16) NOT NULL
     ,
  `entity_type` VARCHAR(16) NULL DEFAULT
      NULL,
  `entity_text` VARCHAR(255) NULL
     DEFAULT NULL,
  PRIMARY KEY (`serial_id`),
  INDEX `EID` (`entity_id` ASC),
  INDEX `ELANG` (`entity_language` ASC),
  INDEX `ETYPE` (`entity_type` ASC),
  INDEX `ETEXT` (`entity_text` ASC))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **serial_id**: Auto incremental serial ID. Since language information is introduced here and entity ID cannot be used as a Primary Key, we use this serial_id here as a Primary Key.

- **entity_id**: ID of the corresponding entity. Eg. Q5, P110, Q123423, etc.

- **entity_language**: Language of the label of the corresponding entity. Eg. Zh-cn, en, etc.

- **entity_type**: The entity type identifier. ”item” for data items, and ”property” for properties.

- **entity_text**: Contains the labels in different languages. Eg. square kilometre, Kilometro quadrato, Vierkante kilometer, etc.

Indexed tables: besides indexing the Primary Key serial_id, other keys including entity_id, entity_language and entity_type are also indexed.

2.2 description

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    description` (
  `serial_id` BIGINT(32) NOT NULL
     AUTO_INCREMENT,
  `entity_id` VARCHAR(32) NOT NULL,
  `desc_language` VARCHAR(8) NULL,
  `desc_text` VARCHAR(255) NULL,
  PRIMARY KEY (`serial_id`),
  INDEX `EID` (`entity_id` ASC),
  INDEX `DLANG` (`desc_language` ASC))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **serial_id**: Same as in table ’entity’.

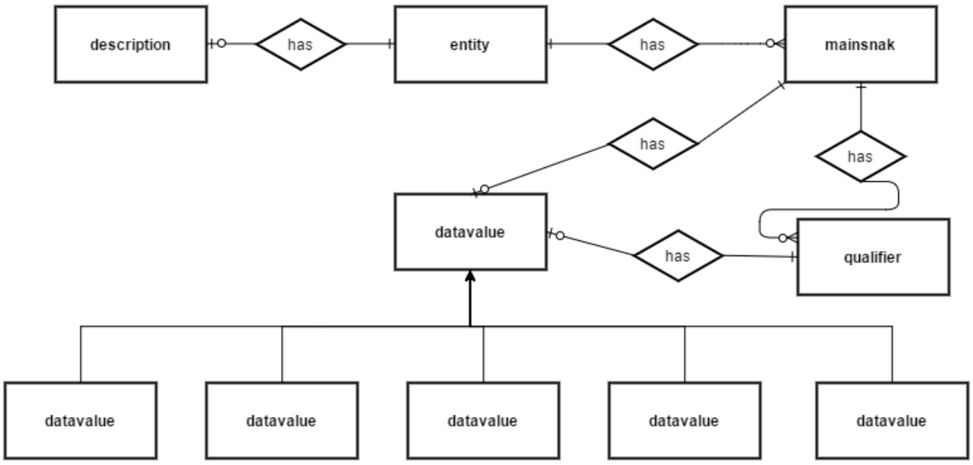


Figure 1: The ER diagram of our design

- **entity_id**: Same as in table ’entity’.
- **desc_language**: Similar to entity_language in table ’entity’.
- **desc_text**: Similar to entity_text in table ’entity’.

Indexed tables: besides indexing the Primary Key serial_id, other keys including entity_id, desc_language and are also indexed.

2.3 mainsnak

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    mainsnak` (
  `snak_id` VARCHAR(64) NOT NULL,
  `entity_id` VARCHAR(32) NOT NULL,
  `property_id` VARCHAR(32) NOT NULL,
  `serial` INT(4) NOT NULL,
  `claimtype` VARCHAR(32) NULL DEFAULT
      NULL,
  `snaktype` VARCHAR(32) NULL DEFAULT
      NULL,
  `datatype` VARCHAR(32) NULL DEFAULT
      NULL,
  `rank` VARCHAR(32) NULL DEFAULT NULL,
  PRIMARY KEY (`snak_id`),
  INDEX `EID` (`entity_id` ASC),
  INDEX `PID` (`property_id` ASC),
  INDEX `CTYPE` (`claimtype` ASC),
  INDEX `STYPE` (`snaktype` ASC),
  INDEX `DTYPE` (`datatype` ASC))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **snak_id**: An arbitrary identifier for the claim, used as Primary Key of this table.
- **entity_id**: ID of the entity this claim belongs to.
- **property_id**: The property this claim is describing.
- **serial**: This field is X means this is the Xth claim of this property.
- **claimtype**: The type of the claim - currently either statement or claim.
- **snaktype**: The type of the snak. Currently, this is one of value, somevalue or novalue.
- **datatype**: The datatype field indicates how the value of the Snak can be interpreted. This fields indicate the concrete table begining with ”datavalue_”
- **rank**: The rank expresses whether this value will be used in queries, and shown be visible per default on a client system. The value is either preferred, normal, or deprecated.

2.4 datavalue_string

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    datavalue_string` (
  `snak_id` VARCHAR(64) NOT NULL,
  `value` VARCHAR(255) NULL DEFAULT NULL
      ,
  PRIMARY KEY (`snak_id`))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **snak_id**: Primary key. Refering to the table mainsnak.

- **value**: The content of the string.

2.5 datavalue_time

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    datavalue_time` (
  `snak_id` VARCHAR(64) NOT NULL,
  `time` VARCHAR(64) NULL DEFAULT NULL,
  `timezone` VARCHAR(32) NULL DEFAULT
     NULL,
  `before` VARCHAR(32) NULL DEFAULT NULL
     ,
  `after` VARCHAR(32) NULL DEFAULT NULL,
  `precision` INT(8) NULL DEFAULT NULL,
  `calendarmodel` VARCHAR(255) NULL
     DEFAULT NULL,
  PRIMARY KEY (`snak_id`))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **snak_id**: Primary key. Refering to the table mainsnak.
- **time**: The content of the string.
- **timezone**: Signed integer. Now unused.
- **before**: Begin of an uncertainty range, given in the unit defined by the precision field. This cannot be used to represent a duration.
- **after**: End of an uncertainty range, given in the unit defined by the precision field. This cannot be used to represent a duration.
- **precision**: To what unit is the given date/time significant.
- **calendarmodel**: A URI of a calendar model, such as gregorian or julian. Typically given as the URI of a data item on the repository.

2.6 datavalue_globecoordinate

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    datavalue_globecoordinate` (
  `snak_id` VARCHAR(64) NOT NULL,
  `latitude` FLOAT NULL DEFAULT NULL,
  `longitude` FLOAT NULL DEFAULT NULL,
  `altitude` FLOAT NULL DEFAULT NULL,
  `precision` FLOAT NULL DEFAULT NULL,
  `globe` VARCHAR(255) NULL DEFAULT NULL
     ,
  PRIMARY KEY (`snak_id`))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **snak_id**: Primary key. Refering to the table mainsnak.
- **latitude**: The latitude part of the coordinate in degrees, as a float literal (or an equivalent string).
- **longitude**: The longitude part of the coordinate in degrees, as a float literal (or an equivalent string).
- **altitude**: Deprecated and no longer used. Will be dropped in the future.
- **precision**: the coordinate's precision, in (fractions of) degrees, given as a float literal (or an equivalent string).
- **precision**: To what unit is the given date/time significant.
- **globe**: the URI of a reference globe.

2.7 datavalue_quantity

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    datavalue_quantity` (
  `snak_id` VARCHAR(64) NOT NULL,
  `amount` VARCHAR(64) NULL DEFAULT NULL
     ,
  `upperBound` VARCHAR(64) NULL DEFAULT
     NULL,
  `lowerBound` VARCHAR(64) NULL DEFAULT
     NULL,
  `unit` VARCHAR(64) NULL DEFAULT NULL,
  PRIMARY KEY (`snak_id`))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **snak_id**: Primary key. Refering to the table mainsnak.
- **amount**: The nominal value of the quantity, as an arbitrary precision decimal string. The string always starts with a character indicating the sign of the value, either "+" or "-".
- **upperBound**: Optionally, the upper bound of the quantity's uncertainty interval, using the same notation as the amount field.

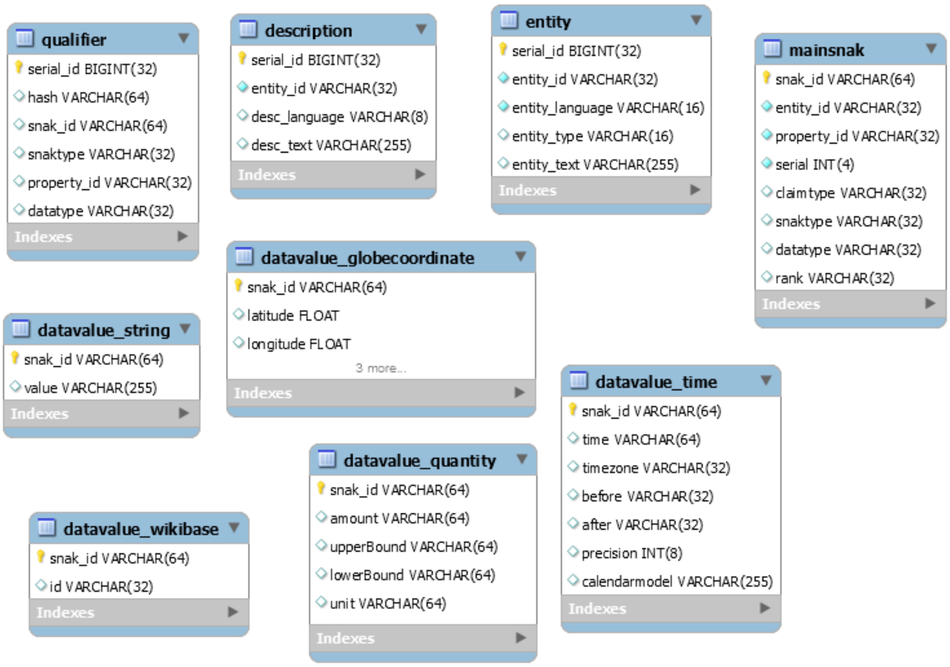


Figure 2: Details of our table design

- **lowerBound**: Optionally, the lower bound of the quantity's uncertainty interval, using the same notation as the amount field.

- **unit**: the URI of a unit (or ”1” to indicate a unit-less quantity).

2.8 datavalue_wikibase

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    datavalue_wikibase` (
  `snak_id` VARCHAR(64) NOT NULL,
  `id` VARCHAR(32) NULL DEFAULT NULL,
  PRIMARY KEY (`snak_id`))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **snak_id**: Primary key. Refering to the table mainsnak.

- **id**: defines the id of the entity.

2.9 qualifier

Table schema:

```
CREATE TABLE IF NOT EXISTS `wikidata`.`
    qualifier` (
  `serial_id` BIGINT(32) NOT NULL
    AUTO_INCREMENT,
  `hash` VARCHAR(64) NULL DEFAULT NULL,
  `snak_id` VARCHAR(64) NULL DEFAULT
    NULL,
  `snaktype` VARCHAR(32) NULL DEFAULT
    NULL,
  `property_id` VARCHAR(32) NULL DEFAULT
    NULL,
  `datatype` VARCHAR(32) NULL DEFAULT
    NULL,
  INDEX `STYPE` (`snaktype` ASC),
  INDEX `PID` (`property_id` ASC),
  INDEX `DTYPE` (`datatype` ASC),
  PRIMARY KEY (`serial_id`),
  INDEX `SID` (`snak_id` ASC),
  INDEX `HASH` (`hash` ASC))
ENGINE = InnoDB
```

The data stored in corresponding fields:

- **serial_id**: Primary key.

- **hash**: Hash value of qualifier. Used to build connection between table qualifier and datavalue_ series.

- **snak_id**: The id of the snak this qualifier belongs to. Used to build connection between table qualifier and mainsnak.

- **snaktype**: Similar to the table mainsnak.

- **property_id**: Similar to the table mainsnak.

- **datatype**: Similar to the table mainsnak.

In Figure 2

3 DATA POPULATING

Populating the database - Python script:

In order to populate the database, we have written a Python script to read the JSON file line by line and extract the information that we need. The populating process is as follows:

First we extract the entity ID and entity type, followed by the various languages supported, along with their corresponding descriptions. Once we have this information, we are able to populate both the entity and description tables. Using PyMySQL, we can connect to the MySQL database and effectively insert records. This is followed by the iteration of each property within the claims to populate the remaining tables. We extract information from the mainsnak and qualifiers to insert into the mainsnak table and qualifier table respectively.

Then, we deal with the various types of data values that we may encounter in the mainsnaks and qualifiers. This includes string, wikibase-entityid, time, globecoordinate, and quantity types. For each type, we extract the necessary fields needed to populate the corresponding table. We insert these records in the same fashion as before, and continue on to the next iteration.

For details please refer to wikidataDB.py in our code submission.

4 DATABASE OVERVIEW

Due to the constraint of disk size, we only use 100k entities as the set for our experiment. The experiment on the next section is based on the subset below.

Overall statistics:

name	count
entity	999999
item	999323
property	676

Table statistics:

table	row number
datavalue_globecoordinate	201934
datavalue_quantity	204015
datavalue_string	2759499
datavalue_time	463418
datavalue_wikibase	4501342
description	6836149
entity	10430881
mainsnak	7720798
qualifier	2759499

5 QUERY DESIGN

5.1 Query 1

5.1.1 Requirement

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

5.1.2 Design

```
SELECT entity.entity_id, entity.
    entity_text, description.desc_text
FROM entity, description
WHERE entity_text=’universe’
AND entity_language=’en’
AND description.desc_language=’en’
AND description.entity_id=entity.
    entity_id
```

5.1.3 Performance

Sampling 5 examples:

serial	entity	time
1	universe	0.016sec
2	dig	0.015sec
3	flower	0.016sec
3	president	0.016sec
3	like ’%mi%’	0.172sec

5.2 Query 2

5.2.1 Requirement

Given an entity, return all preceding categories (instance of and subclass of) it belongs to.

5.2.2 Design

```
SELECT entity.entity_text, entity.
    entity_id, mainsnak.property_id,
    entity_language
FROM entity, mainsnak
WHERE (mainsnak.property_id="P31" or
    mainsnak.property_id="P279")
AND mainsnak.entity_id in (
        SELECT ent.entity_id
    FROM entity as ent
    WHERE ent.entity_text="gas")
AND entity.entity_id in (
        SELECT wkb.id
    FROM datavalue_wikibase as wkb
    WHERE wkb.snak_id=mainsnak.snak_id)
        and entity.entity_language=’en’
```

5.2.3 Performance

Sampling 5 examples:

serial	entity	time
1	universe	0.047sec
2	apple	0.015sec
3	mud	0.016sec
3	fly	0.032sec
3	wind	0.032sec

5.3 Query 3

5.3.1 Requirement

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

5.3.2 Design

```
SELECT wkb1.id AS entity_id, wkb2.id AS
    query_id
FROM (SELECT snak_id, entity_id,
    property_id, datatype FROM mainsnak
    where datatype="wikibase-item") AS
    snak1,
datavalue_wikibase AS wkb1,
(SELECT snak_id, entity_id, property_id,
     datatype FROM mainsnak where
    datatype="wikibase-item") AS snak2,
datavalue_wikibase AS wkb2
where wkb2.id in (
        SELECT distinct ent2.entity_id
            FROM entity AS ent2
    where ent2.entity_text="bench")
AND snak1.snak_id=wkb1.snak_id
AND snak2.snak_id=wkb2.snak_id
AND snak2.datatype="wikibase-item"
AND wkb1.id<>wkb2.id
AND snak1.property_id=snak2.property_id
AND snak1.entity_id=snak2.entity_id
order by wkb2.id
```

5.3.3 Performance

Sampling 5 examples:

serial	entity	time
1	universe	0.031sec
2	car	0.032sec
3	noodle	0.031sec
3	dragon	0.094sec
3	china	3.171sec

5.4 Query 4

5.4.1 Requirement

Given an entity, return all the properties and statements it possesses.

5.4.2 Design

```
-- properties
SELECT e1.entity_text as property_text,
    e1.entity_id as property_id, snak.
    entity_id, des1.desc_text as
    property_desc_text, des2.desc_text
    as desc_text, e2.entity_text
FROM entity as e1, mainsnak as snak,
    description as des1, description as
    des2, entity as e2
WHERE e1.entity_id=snak.property_id
AND e1.entity_language=’en’
AND e2.entity_text="universe" AND e2.
    entity_language=e1.entity_language
AND des2.entity_id=e2.entity_id AND des2
    .desc_language=e1.entity_language
AND snak.entity_id=e2.entity_id
AND des1.entity_id=e1.entity_id AND des1
    .desc_language=e1.entity_language

-- statements
SELECT snak.snak_id, snak.entity_id, des
    .desc_text, ent.entity_text
FROM mainsnak as snak, entity as ent,
    description as des
WHERE snak.entity_id=ent.entity_id
AND ent.entity_text=’gas’ AND ent.
    entity_language=’en’
AND des.entity_id=ent.entity_id AND des.
    desc_language=ent.entity_language
order by entity_id
```

5.4.3 Performance

Sampling 5 examples:

serial	entity	time
1	gas	0.031sec
2	kill	0.000sec
3	fix	0.000sec
3	dare	0.006sec
3	knock	0.016sec

5.5 Q&A Systems

5.5.1 Requirements

Design and implement a basic Q&A (questioning and answering) system, e.g. if I ask what is the population of China, it should return the correct answer.

5.5.2 Code

Please refer to our code submission.

5.6 Extra Queries

5.6.1 Description

Addition to the required queries, we further designed a query to return all the entities as preceding categories of the given entity.

5.6.2 Design

```
select ent.entity_text as res_text, ent.
    entity_id as res_id, property_id
    from entity as ent, mainsnak,
    datavalue_wikibase as wkb
where mainsnak.property_id in (’P31’,’
    P279’)
and wkb.id in (
        select ent1.entity_id from
            entity as ent1
        where entity_text=’physical␣
            system’ and ent1.
            entity_language=ent.
            entity_language)
and ent.entity_language=’en’
and wkb.snak_id=mainsnak.snak_id
and ent.entity_id=mainsnak.entity_id
and mainsnak.datatype=’wikibase-item’
```

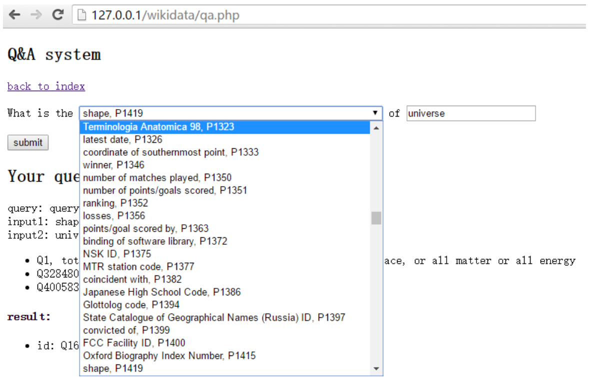
5.6.3 Example

The result of code above is shown in the following table. If we input ”physical system”, one of the result should be ”universe”, since universe is an instance of physical system.

res_text	res_id	property_id
universe	Q1	P31
infrastructure	Q121359	P279
symmetry	Q1455736	P279
optical system	Q2920313	P279

6 USER INTERFACE DESIGN

In this section, we will introduce the design and implementation of our user interface. The user interface is written in PHP. The whole system runs on XAMPP for Windows, which hosts both PHP server and MySQL server. The view of query and Q&A pages are generated by queries.php and qa.php respectively. There is another model.php file, which serves as an interface between the view and the MySQL database.



127.0.0.1/wikidata/qa.php

Q&A system

back to index

What is the shape, P1419 of universe

submit

Your que

query: query
input1: shap
input2: univ

- Q1, tot ace, or all matter or all energy
- Q328480
- Q400583

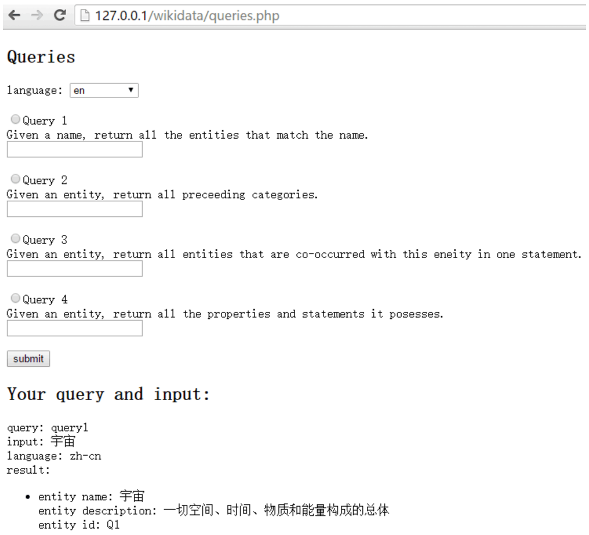
result:

- id: Q16

Terminologia Anatomica 98, P1323
latest date, P1326
coordinate of southernmost point, P1333
winner, P1346
number of matches played, P1350
number of points/goals scored, P1351
ranking, P1352
losses, P1356
points/goal scored by, P1363
binding of software library, P1372
NSK ID, P1375
MTR station code, P1377
coincident with, P1382
Japanese High School Code, P1386
Glottolog code, P1394
State Catalogue of Geographical Names (Russia) ID, P1397
convicted of, P1399
FCC Facility ID, P1400
Oxford Biography Index Number, P1415
shape, P1419

Figure 4: The interface of Q&A system

6.1 Query interface



127.0.0.1/wikidata/queries.php

Queries

language: en

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

○ Query 2
Given an entity, return all preceeding categories.

○ Query 3
Given an entity, return all entities that are co-occurred with this eneity in one statement.

○ Query 4
Given an entity, return all the properties and statements it posesses.

submit

Your query and input:

query: query1
input: 宇宙
language: zh-cn
result:

- entity name: 宇宙
  entity description: 一切空间、时间、物质和能量构成的总体
  entity id: Q1

Figure 3: User interface of query.php

As shown in Figure 3, the first four required queries are integrated into this single page, query.php. The page is consisted of a set of four radio button to select query, corresponding text input, and a menu to select language. To launch any of the queries, the user can just check the desired radio button, and input the entity name. Multi-language is supported, by selecting different languages in the menu.

6.2 Q&A interface

As shown in Figure 4, the Q&A system includes a selection menu and a input area. The user should selects among the properties using the menu, or he/she can also type in this area and the system will automatically find the correct property and fill in. The first input is a property, and the second input is the entity. For example, Figure 4 shows the input when the user wants to ask ”what is the shape of universe”.

Since there may be ambiguity in the second input, we show the possible description of the second input. This can be easily done by query1 mentioned above. The result differs, from quantity to time or entity id.