Re-exam in Databases (TDA357/DIT620)

27 August 2015 at 14:00-18:00, House V

CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Department of Computer Science and Engineering

Examiner: Aarne Ranta tel. 772 10 82. Aarne will visit the exam rooms at around 16:00 and can be reached by mobile phone 0763178590 at other times.

Results: Will be published 11 September at the latest.

Exam review: Individual agreement with the examiner.

Grades: Chalmers: 24 for 3, 36 for 4, 48 for 5. GU: 24 for G, 42 for VG.

Help material: One “cheat sheet”, which is an A4 sheet with hand-written notes. You may write on both sides of that sheet. If you bring a sheet, it must be handed in with your answers to the exam questions. One English language dictionary is also allowed.

Specific instructions:

- You can answer in English, Danish, Dutch, Finnish, French, German, Italian, Norwegian, Spanish, or Swedish (in this exam; next time it can be another set of languages ;-)
- Begin the answer to each question (numbers 1 to 6) on a new page. The a,b,c,... parts with the same number can be on the same page.
- Write clearly: unreadable = wrong!
- Fewer points are given for unnecessarily complicated solutions.
- Indicate clearly if you make any assumptions that are not given in the question.
- In particular: in SQL questions, use standard SQL or Oracle. If you use any other variant (such as MySQL), say this; but full points are not guaranteed since this may change the nature of the question.
- Write the question number on every page. If you need many pages for one question, number the pages as well, for instance, “Question 3 p. 2”.

Question 1: design (12p)

A **dictionary** is a list of words in a language with their properties such as inflection and translation. Dictionaries on a computer are often stored as databases. In this question, your task is to design a database for a dictionary that stores English and Swedish nouns ("substantiv" in Swedish) and their translations. The database should store the following things:

- English nouns in their "dictionary form", also known as **lemmas**. A lemma is the singular form of the noun, for instance, *school*.
- Swedish lemmas, which are singular indefinite forms, such as *skola*.
- For every English noun, also its plural form, for instance *schools* for *school*. The plural of each noun is unique (which is a slight oversimplification).
- For every Swedish noun, also its singular definite (*skolan*), plural indefinite (*skolor*) and plural definite (*skolorna*) forms. These forms are unique for each lemma (again, a slight oversimplification). Every Swedish noun also has a unique gender, whose value is either *real* (corresponding to the article "en") or *neuter* (corresponding to the article "ett").
- The relationship "X translates to Y", where X is an English noun lemma and Y is a Swedish noun lemma. This relationship is many-to-many: for instance, English *school* is both *skola* and *stim* (as in "school of fish") in Swedish, and Swedish *man* is both *man* and *husband* in English.

1a. Draw an E-R diagram for the dictionary database as described above. (7p)

1b. Write a database schema for the database, corresponding to the E-R diagram. (5p)

Don't forget to identify keys and mark them by underlining!

Question 2: normal forms and dependencies (8p)

2a. Give an example of a relation that is in BCNF (Boyce-Codd Normal Form) but not in 4NF (Fourth Normal Form). Show all the information that is needed: attributes, dependencies, keys, etc, clearly stating what the 4NF violations are, as well as an instance (a set of tuples). (5p)

2b. Transform your relation in (2a) to 4NF. (3p)

Question 3: SQL construction and querying (12p)

Still in the dictionary domain, consider another way of storing the words: as a table that relates every word form to its lemma, word class (noun, adjective, verb), and form description (singular definite neuter, infinitive, etc):

- `Words (string, lemma, class, description)`

Thus for instance the Swedish noun form *skolor* could be represented as the tuple

- `('skolor', 'skola', 'noun', 'plural indefinite real')`

3a. Write the database schema in SQL expressing the following constraints:

- the word class is one of `'noun', 'verb', 'adjective'`
- the lemma, class, and description uniquely determine the string (e.g. every noun has a unique plural indefinite form)

(The latter constraint is an oversimplification, but we assume it true in this question.) (3p)

3b. Write an SQL query that returns all noun lemmas with their plural indefinite forms. Notice that 'plural indefinite' is not a complete description, but a part of either 'plural indefinite real' or 'plural indefinite neuter'. (4p)

3c. Write an SQL query that counts the number of occurrences for every class+description pair, sorted in descending order. The output should look as follows:

class	description	occurrences
noun	singular indefinite real	18999
verb	infinitive	12844
...	...	...

saying that there are 18999 rows marked 'noun' and 'singular indefinite real', and so on. (5p)

Question 4: relational algebra (8p)

Continuing with the dictionary relation of the previous question,

- `Words (string, lemma, class, description)`

write a relational algebra query that returns those strings whose class is ambiguous, i.e., can have two or more different values. An example is *läcker*, which is both the present tense of the verb *läcka* ("leak") and the singular real form of the adjective *läcker* ("delicious"). (8p)

Question 5: SQL triggers (12p)

In Questions 3 and 4, we represented the dictionary with the schema

- `Words (string, lemma, class, description)`

One problem with this schema is that it does not guarantee that words have all their forms included: it can for instance happen that a noun has only its indefinite singular included, not the other forms. One way to solve this is to use a different structure, which stores tuples that are complete sets of forms for each word: `('skola','skolan','skolor','skolorna')`; cf. Question 1. We will now do this for Verbs; Nouns and Adjectives would work in a similar way.

5a. To start, create a view `Verbs` on the table `Words`:

- `Verbs (infinitive, present, past, supine)`, putting together the four forms of each verb that have the same lemma (i.e. the infinitive form), e.g. *dricka, dricker, drack, druckit* ("to drink") and the different descriptions.

(4p)

5b. On the view `Verbs`, create a trigger that adds verbs to table `Words`. An insertion to `Verbs` should thus add four rows to `Words`, linking the four forms to one and the same lemma. (8p)

Question 6: indexes (8p)

Now we have seen two ways of storing words:

1. In a single table `Words`, with different forms on different rows.
2. In separate tables `Verbs,Nouns,Adjectives`, with all forms of a word on the same row.

Assume that the table *Words* is stored in 50 disc blocks, and each of the tables `Verbs, Nouns, Adjectives` is stored in 10 disc blocks.

Consider looking up all the analyses (lemma+class+description) of a given string, e.g. 'läcker'.

- What is the cost of looking up one word in both kinds of databases, assuming that no indexes are defined?
- What would the optimal way to store the lexicon for these queries, and how could we optimize it further by using indexes?

Consider looking up all the forms of a given lemma+class, e.g. *läcka+verb*.

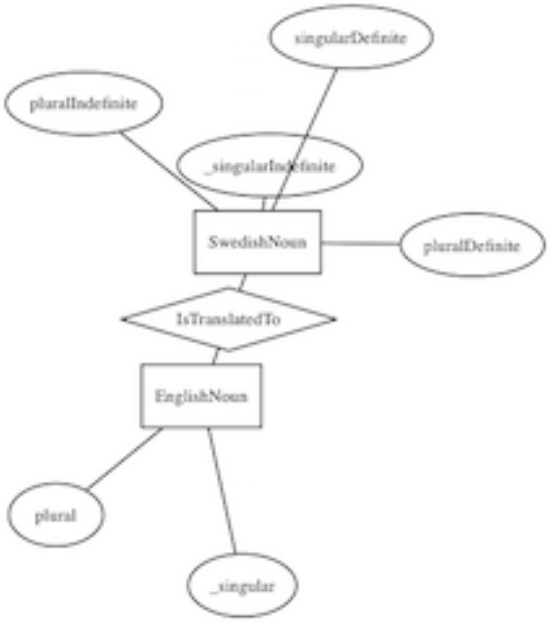
- What is the cost of looking up one word (lemma+class pair) in both kinds of databases, assuming that no indexes are defined?
- What would the optimal way to store the lexicon for these queries, and how could we optimize it further by using indexes?

Databases Re-exam August 2015

Solutions

Aarne Ranta

1 ER diagrams and schemas




```
EnglishNoun(_singular,plural)

SwedishNoun(_singularIndefinite,singularDefinite,pluralIndefinite,pluralDefinite)

IsTranslatedTo(_englishNounSingular,_swedishNounSingularIndefinite)
  englishNounSingular -> EnglishNoun.singular
  swedishNounSingularIndefinite -> SwedishNoun.singularIndefinite
```

2 Functional dependencies and normal forms

From lecture notes, section 4.2.4:

country	product	exportTo
Sweden	cars	Norway
Sweden	paper	Denmark
Sweden	cars	Denmark
Sweden	paper	Norway

```
key (country,product,exportTo)
MVD country ->> product
```

Not in 4NF. Decomposition:

country	product
Sweden	cars
Sweden	paper

```
key (country,product)
```

country	exportTo
Sweden	Norway
Sweden	Denmark

```
key (country,exportTo)
```

3 SQL DDL and Queries

```
-- 3a
CREATE TABLE Words (
  string text,
  lemma text,
  class text,
  description text,
  constraint class_names check (class in ('noun','verb','adjective')),
  constraint words_prim_key primary key (lemma,class,description)
  ) ;

-- 3b

SELECT lemma, string
FROM Words
WHERE description LIKE 'plural indefinite %' ;

-- 3c

SELECT class, description, COUNT(string) AS occurrences
FROM Words
GROUP BY class, description
ORDER BY COUNT(string) DESC ;
```

4 Relational algebra

$$\pi_{A.string}\sigma_{A.string=B.string AND A.class<B.class}(\rho_A words \times \rho_B words)$$

5 Triggers

```
-- 5a

CREATE VIEW Verbs AS (
  SELECT I.string AS infinitive, Pr.string AS present,
            P.string AS past, S.string AS supine
  FROM Words I, Words Pr, Words P, Words S
  WHERE
    I.lemma = Pr.lemma AND I.lemma = P.lemma AND I.lemma = S.lemma AND
    I.description = 'infinitive' AND Pr.description = 'present' AND
    P.description = 'past' AND S.description = 'supine'
    ) ;

-- 5b

CREATE OR REPLACE FUNCTION add_verb() RETURNS trigger AS $$
BEGIN
INSERT INTO Words VALUES(NEW.infinitive, NEW.infinitive, 'verb', 'infinitive') ;
INSERT INTO Words VALUES(NEW.present,    NEW.infinitive, 'verb', 'present') ;
  INSERT INTO Words VALUES(NEW.past,       NEW.infinitive, 'verb', 'past') ;
INSERT INTO Words VALUES(NEW.supine,     NEW.infinitive, 'verb', 'supine') ;
RETURN NEW ;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER AddVerb
  INSTEAD OF INSERT ON Verbs
  FOR EACH ROW
  EXECUTE PROCEDURE add_verb() ;
```

6 Indexes

```
-- all analyses of string

SELECT * FROM Words WHERE string = 'lcker' ;
-- 50; with index on string, 1+k where k is the number of different analyses


WITH Forms AS (
```

```
  SELECT *
  FROM Verbs
  WHERE 'lcker' in (infinitive, present, past, supine)
  )
SELECT 'lcker' as string, infinitive as lemma, 'verb' as class,
           'infinitive' as description
FROM Forms
WHERE infinitive = 'lcker'
  UNION
SELECT 'lcker' as string, infinitive as lemma, 'verb' as class,
            'present' as description
FROM Forms
WHERE present = 'lcker'
  UNION
SELECT 'lcker' as string, infinitive as lemma, 'verb' as class,
            'past' as description
FROM Forms
WHERE past = 'lcker'
  UNION
SELECT 'lcker' as string, infinitive as lemma, 'verb' as class,
            'supine' as description
FROM Forms
WHERE supine = 'lcker'
;
-- UNION the same from nouns and adjectives
-- cost 30 = 10+10+10; with indexes on each form, vf + nf + af,
--   certainly larger than in lookup from Words


-- all forms of lemma+class

SELECT * FROM Verbs WHERE infinitive = 'lcka' ;
-- or Nouns or Adjectives
-- 10; with index on lemma, 2 = 1+k where k=1 by assumption


SELECT description, string
FROM Words
WHERE lemma='lcka'
  AND class='verb' AND description IN ('infinitive','present','past','supine') ;
-- 50; with index on (lemma,class), 2 = 1+k where k=1 by assumption

/*
description | string
------------+--------
 infinitive  | lcka
```

```
 past        | lckte
 present     | lcker
 supine      | lckt

which is good enough for the purpose.

If you want

 infinitive | present |  past  | supine
------------+---------+--------+--------
 lcka       | lcker   | lckte | lckt

expand it to the following, with the same cost:
*/

WITH Forms AS (
  SELECT description, string
  FROM Words
  WHERE lemma='lcka' AND class='verb' AND
    description IN ('infinitive','present','past','supine')
  ),
  Infinitives AS (SELECT string AS infinitive FROM Forms WHERE description = 'infinitive'),
  Presents    AS (SELECT string AS present    FROM Forms WHERE description = 'present'),
  Pasts       AS (SELECT string AS past       FROM Forms WHERE description = 'past'),
  Supines     AS (SELECT string AS supine     FROM Forms WHERE description = 'supine')
SELECT * FROM Infinitives, Presents, Pasts, Supines
;
```