Informatics and Mathematical Modelling
DTU
March 2002

QUEST

Querying a Relational Database by Keywords

An Exercise in Logic Program Construction in Course 02280 Data Logic

1 Introduction

Information in relational databases may be retrieved using dedicated query languages such as SQL or the relational database algebra. However, a drawback of such languages is that the user must be familiar with logical operators and set operations. On the other hand all computer users know how to apply search machines, where querying is done simply with keywords for retrieval of text pages containing the stated keywords. It is therefore tempting to try to achieve simpler forms of relational database retrieval by means of keywords.

2 QUEST

QUEST is a proposal for an "intelligent" relational database query system in which a query takes form of one or more keywords (identifiers), and the result is delivered in the form of (projected) tuples from the database relations.

The idea is that the set of supplied keywords may comprise attribute names as well as the entities contained in the database. It is then up to QUEST to "guess" (induce) what may be the intended meaning of a set of such keywords.

Example: Database relation [1] :

```
courses

courseno coursename  time points
---------------------------------
c2280    datalogic   f2   points10
c2282    artifint    e4B  points5
c2284    knowbased   e4A  points5
c2286    autoagents  f2A  points5
c2288    advanceddb  f2a  points5
```

This database may be queried with the set of keywords

[1] For simplicity's sake all entities in a database relation may be assumed to take form of identifiers, so there is no alternative numerical data type.

1) `time datalogic points`

which is taken to mean that the tuple(s) where 'datalogic' appears are to be selected and projected on 'time' and 'points'. Similarly the query

2) `datalogic`

may be taken to mean that again tuples containing 'datalogic' are to be selected, but now without subsequent projection.

Clearly keyword based queries are potentially ambiguous, since e.g. an identifier may appear simultaneously as attribute as well as in a tuple.

3 A QUEST Prototype

A prototype of the above outlined query system should be constructed in Prolog.

3.1 Representing Data

Here follows hints for how to represent the database.

The database schema may be described with atomic factual clauses

$relations([r_1, ...])$
$schema(r, [a_1, a_2, ..., a_n])$

For each attribute there may be a synonym resolution clause:

$synonyms(id, [id_1, ..., id_k])$

where id is in the second argument list.

The contents of a relation may be given in one factual clause:

$relation(r, [[c_{11}, c_{12}, ..., c_{1n}], ..., [c_{m1}, c_{m2}, ..., c_{mn}]])$

3.2 Calling QUEST

Your QUEST prototype system may be called with goal clauses, say, of the form

$$\leftarrow quest([w_1, w_2, ...], X)$$

where $[w_1, w_2, ...]$ is the list of keywords forming a query, and X is the variable in which the answer to the query is obtained. You may design the system so that each tuple in the QUEST query answer gives rise to one instantiation of X with an appropriate self-describing tuple (list), consisting, say, of attribute-value pairs. Alternatively, you may choose to aggregate the result relation as one list of tuples in X.

3.3 Prototype Architecture

The QUEST system may comprise two major parts:

1. A query-analyser, which analyses the set (list) of keywords and constructs (0, 1 or more) internal query descriptors.
2. A query-evaluator, which evaluates a query descriptor with respect to the database contents.

An internal descriptor is a compound term comprising
1) a relation name
2) An attribute list stating the projection
3) a (possibly empty) list of selection conditions, where a condition consists of an attribute and a value.

You may assume that there is 0 or 1 condition, only.

You may also assume for simplicity's sake that there is just one relation in the database.

3.4 Accessing Relations through Views

Relations may be given virtually as so called database views.

For instance if there is room information (for spring 2002) given as `rooms`:

```
courseno building room
----------------------
c2280    b308     aud12
c2280    b322     room033
c2288    b308     aud13
```

then this relation could be kept as an "internal" one, and made accessible through a virtual relation given by the clause

$$courseswithrom(C, CN, T, P, B, R) \leftarrow courses(C, CN, T, P), rooms(C, B, R).$$

4 Special Prolog Facilities

Formulation of programs in the form of pure logical Prolog (definite clauses) is encouraged. However, you should be aware in particular of the below mentioned extra-logical facilities.

4.1 Negation-as-failure

Negation is not admitted in front of atomic formulae in definite clauses.

There is in Prolog, however, a distingusihed form of negation, negation as failure (to prove) often written "\+" (emulating $\nvdash$) as used in the clause

```
notmember(X,L) :-  \+ member(X,L).
```

where member is specified as usual.

Here X and L are to be instantiated to ground terms. Thus this negation expresses success iff the succeeding atomic formula considered a goal fails.

In some Prolog systems this negation is written as `not` or `not( ... )`.

4.2 Input/output predicates

There are distinguished input/output predicates. For output there e.g. `write( )` and `nl` (i.e. new line). See further the manual of your Prolog system.

5 Form of Report

The report is expected to take up, say, 6-8 pages excluding appendices. Don't use 8 pages if you think 5 pages is enough. However, don't forget some instructive examples.

You should assume that the reader is familar with logic programming and with relational data bases. However, you have to explain your prototype version of QUEST. Try to make clear the principles behind your prototype program design. Explain the form of applied data structures (use examples) as well as the purpose of the most important predicates you introduce. *It is crucial that the reader find your report to be well-structured and easy to understand.*

Here is a check list for the report:

1. The front page should comprise the following information: Name of the exercise. Full names and study numbers of authors. Date and year. Group number.
2. The report should be divided into appropriate sections at the discretion of the authors. There must be an introductory section and a conclusion. Pages should be numbered. The conclusion should summarise the status of the program system and experiences with use.
3. Program listings and output examples should be put in an appendix. However, pieces of the source program may appear in the main text with appropriate explanations.

Jørgen Fischer Nilsson

quest.tex