Fall, 2004 CIS 550

Database and Information Systems

Midterm Solutions

The exam is 80 minutes long. There are 100 points total, plus 10 points extra credit. Please look at both sides of the paper. The last problem is extra credit and should be attempted last.

Problem 1 [30 points]: Consider the following relational schema representing a database of email messages with a keyword index:

Email(eid:*integer*, from:*string*, to:*string*, date:*string*, subject:*string*, body:*string*)
Occurrence(kid:*integer*, eid:*integer*, keyword:*string*, position:*integer*)

For each of the following queries, translate the query to (i) SQL, (ii) the relational algebra, if it can be expressed that way, and (iii) the domain relational calculus, if it can be expressed that way.

1. Find all documents older than 2004-12-01. (Assume that dates can be compared using lexicographic string comparison.)

   Answer:

   (i)
   ```
    SELECT *
   FROM Email
   WHERE date < '2004-12-01';
   ```

   (ii) $\sigma_{\text{date}<\text{`2004-12-01'}}(\text{Email})$

   (iii) $\{\langle e,f,t,d,s,b\rangle \mid \langle e,f,t,d,s,b\rangle \in \text{Email} \wedge d < \text{`}2004-12-01'\}$

2. Find all keywords occurring within 10 positions of “Oracle” in the same document.

   Answer:

   (i)
```
SELECT o1.keyword
FROM Occurrence o1, Occurrence o2
WHERE o2.keyword = 'Oracle' AND o1.eid = o2.eid AND
o1.position - o2.position >= -10 AND o1.position - o2.position <= 10;
```

   (ii) $\pi_{\text{keyword}}(\sigma_{|p_1-p_2|\leq 10}(\rho_{\text{position}\mapsto p_1}(\text{Occurrence}) \bowtie \pi_{\text{eid,position}}(\sigma_{\text{keyword}=\text{`Oracle'}}(\rho_{\text{position}\mapsto p_2}(\text{Occurrence})))))$

   (iii) $\{\langle w_1\rangle \mid (\exists k_1, k_2, e, p_1, p_2)(\langle k_1, e, w_1, p_1\rangle \in \text{Occurrence} \wedge \langle k_2, e, \text{`Oracle'}, p_2\rangle \in \text{Occurrence} \wedge |p_1 - p_2| \leq 10)\}$

3. Find the word most commonly occurring with “Oracle”.

   Answer:

   (i)
```
SELECT o1.keyword
FROM Occurrence o1, Occurrence o2
WHERE o1.eid IN (
SELECT DISTINCT o2.eid
FROM Occurrence o2
WHERE o2.keyword = 'Oracle'
)
GROUP BY o1.keyword
HAVING COUNT(o1.keyword) >= ALL (
SELECT COUNT(o3.keyword)
FROM Occurrence o3 WHERE o3.eid IN (
SELECT DISTINCT o4.eid
FROM Occurrence o4
WHERE o4.keyword = 'Oracle'
)
GROUP BY o3.keyword
);
```

   (ii) Can’t be expressed

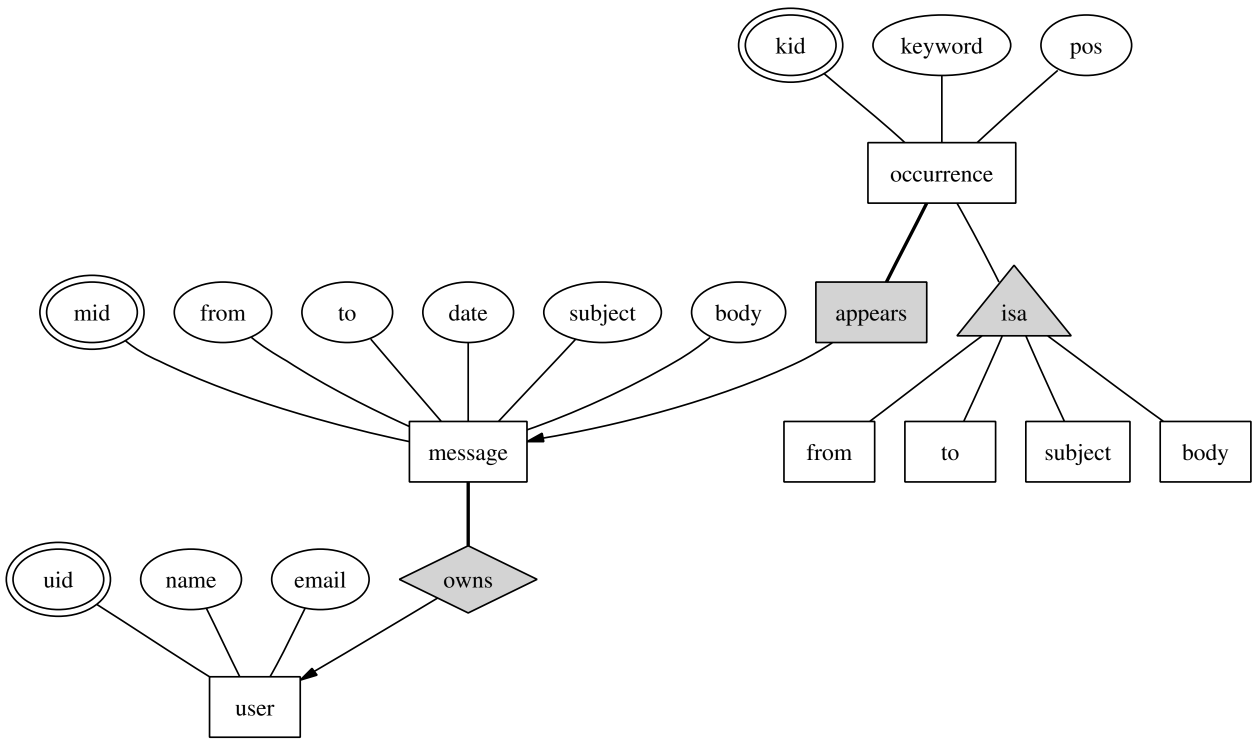
   (iii) Can’t be expressed

Problem 2 [30 points]: Suppose we would like to extend the schema from the previous question to represent the following situation:

- An email *message* has a unique identifier, a from field, a to field, a date field, a subject field, and a body field.
- A *user* has a unique identifier, a name field, and an email address field.
- Each email message corresponds to exactly one user.
- A keyword *occurrence* has an identifier, a keyword and a position.
- Additionally, every occurrence is either in: the from field; the to field; the subject field; or the body field.
- Each occurrence corresponds to exactly one message.

1. Draw an ER-diagram satisfying these requirements. Specify any covering or overlap constraints.

   Answer:



`from AND to AND subject AND body COVER occurrence`

2. Write down the corresponding relational schema, normalized to 3NF.

   Answer:
   Email(eid:*integer*, from:*string*, to:*string*, date:*string*, subject:*string*, body:*string*)
   User(uid:*integer*, name:*string*, email:*string*)
   Occurrence(kid:*integer*, keyword:*string*, position:*integer*)
   Owns(uid:*integer*,mid:*integer*)
   Appears(kid:*integer*,mid:*integer*)
   From(kid:*integer*)
   To(kid:*integer*)
   Subj(kid:*integer*)
   Body(kid:*integer*)

   The only FDs are the key dependencies indicated by underlining. The schema is already in 3NF.

Problem 3 [10 points]: Answer briefly the following question:

What is the difference between a key and a functional dependency?

Answer:
A functional dependency is a rule that says some attributes in a relation determine some other attributes in the relation. In the special case where all attributes in the relation are determined, we call the determining attributes a *key*. Formally, given a relation R on attributes U, and a set of functional dependencies F, we say that a set $X \subseteq U$ is a *superkey* for R iff $X \rightarrow U \in F^+$, the closure set of F. If, in addition, X is minimal (i.e. there is no $Y \subset X$ s.t. $Y \rightarrow U$), then we call X a *key*.

Problem 4 [30 points]: Consider the following fragment of the XML document `guide.xml` representing an online city guide:

```
<cityguide>
  <city key="phila">
    <name> Philadelphia </name>
    <country> USA </name>
    <lang> en </lang>
  </city>
  ...
  <restaurant>
    <name> Jim's Steaks </name>
    <number> 400 </number>
    <street> South St </street>
    <cuisine> American </cuisine>
    <rating> 5 </rating>
    <description> A memorable cheese steak experience. </description>
    <city> phila </city>
  </restaurant>
  ...
</cityguide>
```

Translate the following queries to XQuery, nesting the results inside an `<answer>` tag:

1. Find the names of all restaurants serving American cuisine.

   Answer:
   ```
   <answer> {
   for $r in document("guide.xml")/cityguide/restaurant
   where $r/cuisine = "American"
   return $r/name
   } </answer>
   ```

2. Find the number of restaurants on South St.

   Answer:
   ```
   <answer> {
   let $c := fn:count(document("guide.xml")/cityguide/restaurant [/street/text()
   = "South St"])
   return $c
   } </answer>
   ```

3. Find the names of all restaurants in the USA with a rating of at least 5.

   Answer:
   ```
   <answer> {
   for $r in document("guide.xml")/cityguide/restaurant,
   $c in document("guide.xml")/cityguide/city
   where $r/rating >= 5 and $r/city = $c/@key and $r/country = "USA"
   return $r/name
   } </answer>
   ```

Problem 5 [10 extra credit points]: Consider the subset of XQuery containing simple FLWOR expressions, but no aggregate functions or user-defined functions.

1. Propose a simple logical programming language for this subset of XQuery, analogous to the domain relational calculus. *Hint:* Observe that an XQuery FLWOR block is very similar to a relational select-project-join block, except:

   (a) Variables are bound based on path expressions (which may require new primitives),

   (b) Query block output should be a collection of trees with bound variables within tags, rather than a collection of tuples.

   Answer:
   The variables in our language are bound to *trees* or *sets* of trees. We allow XPath expressions as primitives in the language, which we interpret as functions from *trees* to *sets* of trees, in the usual way. We also take as a primitive the *document* function. For simplicity, we ignore order. A query result is a *set* of trees. The head of the expression in our language is a template corresponding to the XQuery return clause, composed of a variable or list of variables, possibly nested inside XML tags. For example, return <answer > $t </answer > would map to $\{$ <answer > $\$t$ </answer > $| \ldots\}$. The body of the expression contains quantifiers and conditions corresponding to the quantifiers and conditions in the XQuery for, let, where clauses. For example, for $p in document("db-inproc.xml")/dblp/inproceedings translates to $\{\ldots \mid (\exists p)(p \in$ document("db − inproc.xml")/dblp/inproceedings$)\}$.

2. Illustrate your language by translating the following query to it:

```
for $p in document('dblp-inproc.xml')/dblp/inproceedings,
    $a in $p/author
let $t := $p/title
where $a = 'Serge Abiteboul'
return $t;
```

Answer:

$$\begin{array}{lll} \{\ t \mid (\exists p, a, t) & (& p \in \textsf{document('dblp} - \textsf{inproc.xml')/dblp/inproceedings} \\ & \wedge & a \in p/\textsf{author} \\ & \wedge & t = p/\textsf{title} \\ & \wedge & a = \textsf{'Serge Abiteboul'}\ )\ \} \end{array}$$