P8q9
GMB

Answer to ~~Question 1~~ Database Theory 2003

1. *Describe the key features of the ODMG object model.* *[6 marks]*

   [Bookwork] The key component of the ODMG object model is an object. Every object has a unique oid and the database contains collections of objects, called classes. The collections are named using extents (the extent names can be used in OQL queries). Properties of classes are specified using ODL (Object Definition Language), and are of three kinds: attributes, relationships and methods.

   Attributes have either an atomic type (integers, reals, strings, characters, etc.) or a structured type. ODL supports sets, bags, lists, arrays and struct type constructors, along with other more database-specific datatypes such as date and time, etc.

   Relationships have a type that is either a reference to another object (one-to-one) or a collection of references (one-many). A relationship in the ODMG model has a corresponding inverse relationship; which is intuitively the relationship in the other direction.

   Methods are functions supported by objects in that class. They are written in the programming language for which there is a binding. In Version 3.0 of the standard this includes Java, C++ and Smalltalk.

   Classes form inheritance hierarchies; the ODMG model only supports single-inheritance, like Java. Attributes and methods can be redefined in subclasses, again as in Java.

   The intention of the ODMG is to provide database support for persistent objects. Thus the data model support the datatypes typical in modern object-oriented programming languages, along with those necessary for database programming, such as collection types.

2. *Describe the key features of the semi-structured data model.* *[6 marks]*

   [Bookwork] Semistructured data is often described as "schema-less" or "self-describing", in that unlike the relational or object models we do not insist on a schema. That is we have no schema language or data definition language. The primary reason for interest in the semistructured data model is the flexibility that it offers. In semistructured data the schema information as such is contained in the data, and can vary over time and within the database.

   Semistructured data is best represented as graph. Thus a database of semistructured data is a collection of nodes. Each node is either a leaf node or an interior note. Leaf nodes have data values associated with them, which is of some atomic type (integers, strings, etc.). Interior nodes have one or more arcs out. Each arc has a label. We do not typically impose any restrictions on the labels. Thus it is valid to have more than one arc from the same node with the same label. Typically we will distinguish one node as being the root node. Thus, in conclusion, semistructured data is an edge-labelled graph.

   We can define a simple linear syntax for semi-structured data as follows:

```
ComplexValue ::= {label: SSDExp, ..., label: SSDExp}
SSDExp       ::= Value | oid<Value> | oid
Value        ::= atomic | ComplexValue
```

where oid ranges over a set of object identifiers, and atomic ranges over a set of constants. Here's a simple example:

```
{person: &o1{name: "Mary",
             age: 45,
             child: &o2,
             child: &o3},
 person: &o2{name: "John",
             age: 17},
 person: &o3{name:"Jane",
             relatives: {Mother: &o1,
                         Brother: &o2}}
}
```

3. *Consider the following DTD.*

```
<!ELEMENT class   (student)*>
<!ELEMENT student (name, college, course*, (phone|email)>
<!ELEMENT name    (#PCDATA)>
<!ELEMENT college (#PCDATA)>
<!ELEMENT course  (title, year)>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT year    (#PCDATA)>
<!ELEMENT phone   (#PCDATA)>
<!ELEMENT email   (#PCDATA)>
```

This question is not covered directly in the notes, but relies on a small amount of inspiration from the student. The DTD given only consists of PCDATA (strings), alternation (sum types) and repetition (Kleene star). Thus in answering the question the student needs to consider how to handle these datatype constructors.

(a) *Detail how XML data satisfying such a DTD could be stored in a traditional (non-complex value) relational DBMS.* *[5 marks]*

This relies on an insight that is covered in the notes: XML data (and semi-structured data in general) can be represented as an edge-labelled graph. Thus a naive, inefficient, but plausible strategy is to store the edge-labelled graph using two global relations: One `EDGES(A,B,C)` represents an edge `C` between nodes `A` and `B`; the other `DATA(A,V)` represents the PCDATA `V` associated with node `A`.

This approach is clearly inefficient, and in particular results in very inefficient translation of queries.

A number of optimisations reveal themselves. The most obvious is to partition the global `EDGES` table into a number of tables according to the edge value. Inlining of superfluous edges is also useful.

(b) *Detail how XML data satisfying such a DTD could be stored in an ODMG-compliant DBMS.* *[3 marks]*

It is quite routine to read off ODMG class definitions from the DTD. PCDATA is just a string, and sequences can be represented using the ODMG list datatype. The only complication is sum types, but the inspiration here is to notice that this can be represented using inheritance. Thus the DTD can be represented using the following ODMG ODL declarations.

```
class class
{ attribute list(student) students;
};

class student
{ attribute string name;
  attribute string college;
  attribute list(course) courses;
  attribute phoneoremail contact;
};

class course
{ attribute string title;
  attribute string year;
};

class phoneoremail
{};

class phone extends phoneoremail
{ attribute string phone;
};

class email extends phoneoremail
{ attribute string email;
};
```