Weather News

project report for ICS 624: Advanced Data Management,

December 8th, 2006

Table of Contents

1. Introduction....2
1.a. Project Context....2
1.b. Anticipated System Users....2
2. English Queries....2
3. Semantic Data Model....3
3.a. Entities....3
3.b. Attributes....4
- Semantic....4
- Context....5
- Structural....5

4. MDB Implementation Plan....6
4.a. Multimedia Objects....7
4.b. Complex Attribute Types....8
4.c. Functions....9
4.d. Indexes....10
5. SQL Information Queries....10
5.a. SQL Query 1....10
- Query 1 Discussion....10
- Query 1 Optimization....11

5.b. SQL Query 2....11
- Query 2 Discussion....11
- Query 2 Optimization....12

5.c. SQL Query 3....12
- Query 3 Discussion....13
- Query 3 Optimization....13

6. Evaluation of OR-DBMS....13
7. References....14
8. Appendixes....15
8.a. Appendix A: Full Graphic Models....15
8.b. Appendix B: Attribute Lists....16
- Multimedia Objects....16
- Other Entities....16
- Relations....17

8.c. Appendix C: Create Table Definitions....17

1. Introduction

1.a. Project Context

The mandated context for this assignment is to be drawn from some topic area within a news agency. The application area within a news agency that I am using for my project is weather.

Most major news agencies have a department or section that deals with weather. The implementation of weather for on-line news agencies varies, but there are many possibilities for multiple and multimedia usage within this field. As my experience with weather information (and also with creating databases) is quite limited, I have tried to keep my model relatively simple, however, I have tried to include major facets of weather news such as forecasts, statistics, and weather related news articles.

The database will require someone to input daily weather statistics, forecast statistics, and any new media objects, all of which will come from the news agency. It also requires someone to input metadata about the media objects and maintain the controlled vocabulary.

1.b. Anticipated System Users

One main user of the site is a person that wants to know the quality of the weather. These people will use the site often, and will mostly be interested in the current (or near future) weather quality or weather occurrences for specific places. We can refer to this group as general users.

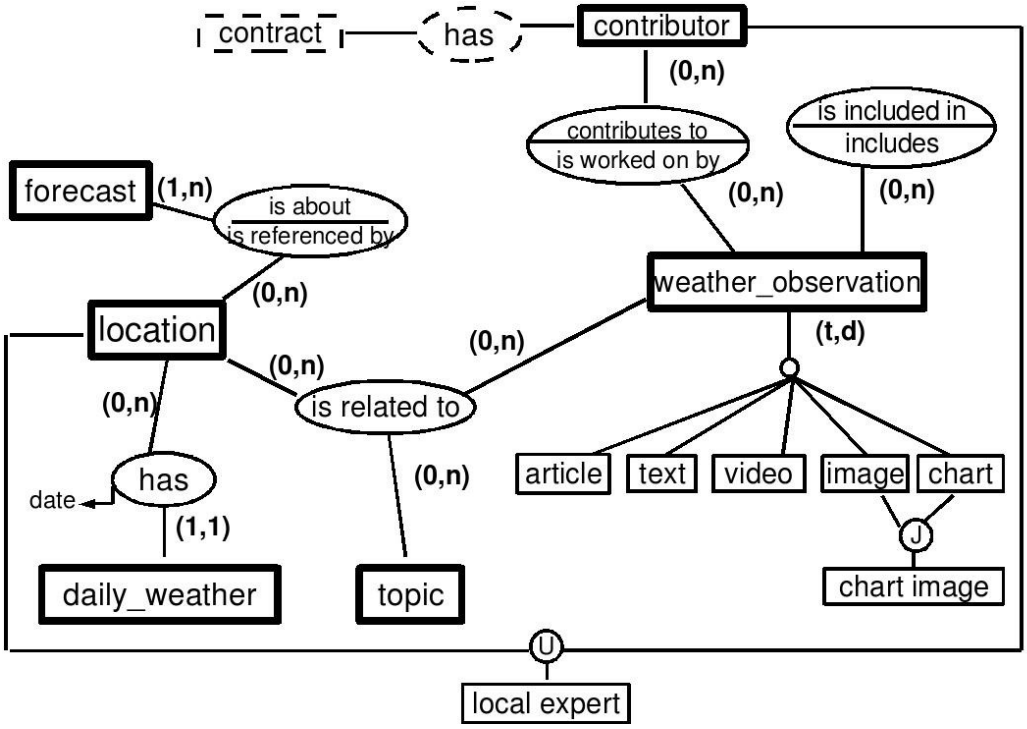
A second user group are persons conducting some sort of research. These people could be looking for weather patterns in different areas, or articles about specific weather occurrences. They might also be looking for specific images or video footage. We can refer to this group as researchers.

2. English Queries

1. What was yesterday's high temperature in Honolulu?
2. Find me a chart showing the average temperature for Atlanta by month?
3. Find news articles about "category 5 hurricanes" in Hawai'i.
4. Find images showing fishing vessels in storms.
5. Show me video footage of a hurricane.
6. Retrieve pictures with the subject storms that appear in articles published after 2004.
7. Retrieve hurricane-related articles with pictures of oceans, along with the dates the articles

were written and the language they were written in.

3. Semantic Data Model



3.a. Entities

Based on the information requirements asrepresented in the English Queries, I determined that the main entities within the system would be locations and what I call weather observations.

In order to be able to store and retrieve weather information about specific locations, I created a **location** entity type (which consists of a city, state, and geographic point). This has an entity type of **daily_weather**, in which will be recorded daily weather statistics. This will function as an archival database; there will be a separate entry for each day to make for easy manipulation of statistical information. The entity type **location** will also have a binary relationship with the entity type **forecast**, each of which might provide forecasts for multiple cities for multiple days.

The entity type **weather_observation** includes both articles (a webpage written in PHP that

includes calls to multiple media objects), as well as the individual components (ie. image and text) that make up the article. There is a classification hierarchy in which the different media types inherit the attributes and methods defined for the superclass **weather_observation**. The participation constraint is total and overlapping, as **image** and **chart** will form a join operation to create the shared subclass **chart_image**. The classification hierarchy allows searching for (and reuse of) specific types of images or video. The multiple media articles (the entity type **article**) are able to reference their various media components by use of the unary relationship **includes**.

In addition to **location** and **weather_observation** there is also the entity type **topic**. This will be a selected and defined set of controlled vocabulary that can be used to describe weather observations. Those three entity types will be related in a ternary associative relationship, in which multiple locations or topics can be assigned to different multiple media articles or individual media components.

There is also the entity type **contributor**. This refers to people who are responsible for the creation of the media objects (ie. authors, photographers, meteorologists), and is related to **weather_observation** in a binary relationship. The **contributor** type has a weak entity relationship to **contract**, which refers to the contract the contributor has with the news agency.

Along with **location**, **contributor** is the parent super-class of of the category entity type **local_expert**. **local_expert** represents a **contributor** to the news agency that has been designated the expert in a specific location.

3.b. Attributes

The basis for the metadata of the media objects is a set of slightly modified Dublin Core elements, which are represented in the model as follows.

Semantic (metadata that characterize the subject matter of the document)

- **dc.Title**
  Represented as an attribute "title" of the entity type **weather_observation**.
- **dc.Subject**
  Represented by the independent entity type **topic** and the associative relationship type **is_related_to**.
- **dc.Description**
  Represented as an attribute "summary" of the entity type **weather_observation**.
- **dc.Coverage**

Represented by the independent entity type **location** and the associative relationship type **is_related_to**.

Context (metadata that describe relationships to external objects)

- **dc.Contributor**
  Represented as an entity **contributor** and the associative relationship type **contributes_to**.
- **dc.Date**
  Represented as an attribute "date" of the entity type **weather_observation**.
- **dc.Relation**
  Represented by the associative relationship type **includes** and the entity type **weather_observation**.

Structural (metadata that describe the internal structure and presentation layout for the media object)

- **dc.Format**
  Represented as an attribute "format" of the entity type **weather_observation**.
- **dc.Language**
  Represented as an attribute "language" of entity type **text**.
- **dc.Identifier**
  Represented as an attribute "id" of the entity type **weather_observation**.

Keywords will be extracted from the "text" attribute of the **text** entity type, along with words from the "title" and "summary" in order to enable full text search. In addition, there will be a text_index of keywords created on all weather observations on the attributes "title" and "summary". Attributes such as publisher are not included, as all of the articles will come from inside the news agency. The contributor attribute is used for instances when credit needs to be given, and will subsume the usual usage of the Dublin Core element *creator*.

A note about the **article** entity. This will include a <text> attribute called "webpage". This will be a dynamic webpage written in PHP, which is a server side scripting language that allows the inclusion of SQL queries. Thus, an article's webpage will include SQL queries to the individual media components that constitute it and have been specified in the **includes** relationship.

4. MDB Implementation Plan

The media objects will have a hierarchy, with the different types of objects inheriting attributes from **weather_observation**. The participation will be total, as all weather observations will be a subclass entity type—either the composite article (**article**) or one of the component parts (**text**, **image**, **video**, **chart**, **chart_image**). Each subclass entity will inherent attributes from **weather_observation**, along with attributes for the specific type ("webpage" for **article**; "text" for **text**, "image" for **image**, etc.). The composite articles will be related to their component parts through the relationship **includes**. Each row in the relationship table will have an **article** id number and the id of number of a media entity that it is related to. This structure will help with locating articles that satisfy queries that call for specific attributes on one of the included media entities during the information retrieval stage, as well as enabling searching that returns individual media objects themselves.

After developing my information requirements, I concluded that location is another one of the primary entities in a weather system. To keep it simple, the **location** entity will have as attributes a geographical point, a city name, and a state name. Treating topic as an entity enables me to develop a controlled vocabulary of common weather subjects and terms that will make categorizing and searching easier.

The is_related_to relationship, which connects a media object to related topics and locations, along with the relevant attribute indexes, will help with the majority of queries that I envision. This will be implemented as two tables, **isrelatedto_location** and **isrelatedto_topic**. There will also be term indexes on the media objects, with only text indexed and searchable by content.

Standard attribute indexes

- unique index on **isrelatedto_location** (weather_observation.id) to facilitate access to particular text objects related to particular locations.
- unique index on **isrelatedto_topic** (weather_observation.id) to facilitate access to particular text objects related to particular topics.
- cluster index on **isrelatedto_location** (location) to facilitate access to locations related to particular text objects.
- cluster index on **isrelatedto_topic** (topic.name) to facilitate access to topics related to particular text objects.
- cluster index on **ContributesTo** (contributor.id) to facilitate access to media objects worked on by particular contributors.

I could also include cluster indexes on various attributes of the media object, including date, contributor, language, and so on, to facilitate access to objects with a specific date, contributor,

language, etc.

I also want users to be able to search by title, summary, and (full) text where available, on the entity type **weather_observation**.

PostgreSQL includeds a full text search module called tsearch2. This will be used to create a weighted index on the attributes title and summary on the relation **weather_observation**, as well as another weighted index on the attributes title, summary, and text of the relation **text.** The search module allows for searching within a specific field, or within multiple fields, with a ranking function to determine the relevance of the query.

4.a. Multimedia Objects

The following statement creates the root entity type **weather_observation** and creates attributes that will be inherited by all of the multimedia objects. Each object will have an id number, a title, summary, format, date, and text_index. The primary key will be the id. This follows the Data Model Translation as laid out in the Nordbotten text in Box 3.2.

```
CREATE TABLE weather_observation
(
  id serial NOT NULL,
  title varchar NOT NULL,
  summary varchar,
  format varchar,
  date date,
  text_index tsvector,
  CONSTRAINT wo_pkey PRIMARY KEY (id)
);
```

The following statement creates a relation for the sub-entity **text** and demonstrates the implementation of hierarchical relations. The INHERITS clause will give the table attributes from the parent weather_observation. The primary key will be created on id. The sub-entity has the additional attributes of text (the full-text object), language (the language the text is written in), and ftindex (a weighted location index on title, summary, and text).

```
CREATE TABLE text
(
  text text,
  language varchar,
  ftindex tsvector,
  CONSTRAINT text_pkey PRIMARY KEY (id)
) INHERITS (weather_observation);
```

The following statement creates the relation **includes**. This table relates the multimedia objects to each other. This is defined by copying the primary keys from article and the weather_observaton it includes and defining them as a composite primary key for the relation. It then defines both as foreign keys using on delete restrict.

```
CREATE TABLE includes
(
  aid int4 NOT NULL,
  mid int4 NOT NULL,
  CONSTRAINT inc_pkey PRIMARY KEY (aid, mid)
  CONSTRAINT inc_art_fkey FOREIGN KEY (aid)
     REFERENCES article (id) MATCH SIMPLE
     ON UPDATE RESTRICT ON DELETE RESTRICT
  CONSTRAINT inc_med_fkey FOREIGN KEY (mid)
     REFERENCES weather_observation (id) MATCH SIMPLE
     ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

4.b. Complex Attribute Types

This creates a new data type FULLNAME that is a composite type of base data types varchar and varchar. This user defined data type will be used in the contributor (and local expert) relations.

```
CREATE TYPE fullname AS
```

("first" varchar,
"last" varchar);

This creates a new data type RESULTS that is a composite type of base data types int, text, and real. This uses two functions that are included in the PostgreSQL tsearch2 module, headline (displays an excerpt including the search term) and rank (numerically ranks the search), and will be useful for returning results from an information query.

CREATE TYPE results AS (id INTEGER, headline TEXT, rank REAL);

4.c. Functions

The following statement creates a trigger that will update the index whenever a new row is added.

CREATE TRIGGER tsvectorupdate BEFORE UPDATE OR INSERT ON text
FOR EACH ROW EXECUTE PROCEDURE tsearch2(ftindex, text);

The following creates a function that takes a text input such as hurricane and uses the full text index to output search results that contain that term(s), ranked by relevance. It utilizes the user defined composite data type "results" that was defined above.

CREATE FUNCTION findobjct(text) RETURNS SETOF results LANGUAGE sql AS '
SELECT id, headline(text, q), rank(ftindex, q)
FROM text, to_tsquery($1) AS q
WHERE ftindex @@ q ORDER BY rank(ftindex, q) DESC';

The following creates a function that automatically inserts values into the relation text, updating the text index vector and assigning weights.

CREATE OR REPLACE FUNCTION insert(varchar, varchar, text) RETURNS void LANGUAGE sql AS
'INSERT INTO text (title, summary, text, text_index)
VALUES ($1, $2, $3, setweight(to_tsvector($1), "B") || setweight(to_tsvector($2), "C") ||

to_tsvector($3));';

4.d. Indexes

The following creates an index on the tableTEXT using the full text index column.

CREATE INDEX ftindex_indx ON text USING gist(ftindex);

In PostgreSQL, CREATE TABLE / PRIMARY KEY/ FOREIGN KEY will create an implicit index on the table's primary key or foreign key.

5. SQL Information Queries

5.a. SQL Query 1

Find news articles about "category 5 hurricanes" in Hawai'i.

SELECT a.id, a.title, a.webpage, headline(t.text, q), rank(t.ft_index, q) AS rank

FROM article a, text t, location l, isrelatedto_location r, includes i, to_tsquery('category & 5 & hurricane') AS q

WHERE l.state = 'Hawaii' AND r.resource = a.id AND r.location = l.id

AND i.aid = a.id AND i.mid = t.id AND t.ft_index @@ q

AND t.text ~* 'category 5 hurricane'

ORDER BY rank DESC;

Query 1 Discussion

This query utilizes classification hierarchies and the SQL3 ability to access hierarchic structures and know that the subclass entities (article, text, image) inherit attributes from the superclass (weather_observation), which is not even mentioned in the query. It also uses the dot notation to specify access paths.

The query also requires utilizing user defined types and functions (or in this case, types and

functions defined by the tsearch2 team and included in postgreSQL as a module). These appear in the SELECT, FROM, and WHERE clauses. The type is tsvector and this is a set of terms that appear within a given text and have been parsed through a dictionary and stemmer. The function to_tsquery is used to query a field with type tsvector. The rank function then uses the function and the type to produce a relevance ranking for a search term.

The query first performs a relatively much faster search on the full text index and then, with the narrowed result set searches the text for the desired string.

Query 1 Optimization

- An optimized execution plan would begin with the clause l.state = 'Hawaii' as it performs an operation on a single table.
- It would then execute the clauset.ft_index @@ to_tsquery('category & 5 & hurricane') as this is another operation eliminating rows from a single table.
- Next would be the join operations to further reduce the result set. This would be the clauses r.resource = a.id AND r.location = l.id and i.aid = a.id AND i.mid = t.id.
- Then, once the set is reduced, execute the query on text, t.text ~* '...', since it is a query on media data that can be time consuming.
- Next the rank and headline functions as they can also be time consuming.
- Finally, order the result set by rank.

5.b. SQL Query 2

Retrieve pictures with the subject storms that appear in articles published after 2004.

```
SELECT i.id, i.title, i.image, a.title, a.date
FROM image i, topic t, isrelatedto_topic r, article a, includes n
WHERE t.name = 'storms' AND t.name = r.topic AND i.id = r.resource
AND a.date > '12-31-2004' AND  a.id = n.aid AND i.id = n.mid
ORDER BY a.date DESC;
```

Query 2 Discussion

This query also makes use of classification hierarchies and inheritance. It also utilizes the indexes automatically created with the creation of primary and foreign keys. The unique thing about this query is that data-type and storage support for unstructured large objects is required by the query. Along with the other attributes, it returns i.image, an image which is an unstructured large object. In postgreSQL, the recommended data type for unstructured large binary objects is bytea.

Query 2 Optimization

- The optimal clauses to start with in this query are the clauses t.name='storms' and a.date > '12-31-2004', as these are select operations on single tables and will eliminate unnecessary rows from the result set.
- Next, the join clauses, t.name = r.topic AND i.id = r.resource and a.id = n.aid AND i.id = n.mid, to further reduce the result set.
- Finally, order by date to prepare the result set for presentation.

5.c. SQL Query 3

Retrieve articles that discuss Hurricane Katrina that include pictures of oceans, along with the dates the articles were written and the language they were written in.

```
SELECT tquery.id, a.title, a.webpage, a.date, tquery.language, sum(tquery.rank + iquery.rank) as
rank
FROM article a,
(SELECT a.id, t.language, rank(t.ft_index, q) as rank
  FROM article a, text t, includes, to_tsquery('hurricane & Katrina') AS q
  WHERE includes.mid = t.id AND includes.aid = a.id
    AND t.ft_index @@ q AND t.text ~* 'Hurricane Katrina') as tquery
INNER JOIN
(SELECT a.id, rank(i.text_index, q) as rank
  FROM article a, image i, includes, to_tsquery('ocean') AS q
  WHERE includes.mid = i.id AND includes.aid = a.id
    AND i.text_index @@ q) as iquery USING (id)
WHERE a.id = tquery.id
GROUP BY tquery.id, a.title, a.webpage, a.date, tquery.language
```

```
ORDER BY rank DESC, a.date DESC
;
```

Query 3 Discussion

This is the most complex of my SQL statements. It selects a set of articles that includes text containing hurricane and Katrina and then performs an inner join on the article id with a set of articles that includes images about oceans. During this, it pulls out information from the different entities involved to satisfy the query. It takes a rank from both sub-select statements and then adds them together in the final select using the sum() function within the statement.

As with Query 1, this query entails the ability to access hierarchical structures, user defined types, and user defined functions.

Query 3 Optimization

- First, the operations in the first sub-select statement of the ft_index to return text rows that contain hurricane.
- Next, the clause includes.mid = t.id AND includes.aid = a.id to reduce this SELECT set to articles containing text that includes hurricane.
- Then the text_index clause on the second sub-select to limit the rows in the image table to those that contain ocean.
- This should be followed by the includes.mid = i.id AND includes.aid = a.id clause to limit this sub-select to articles that contain images about oceans.
- Next, the inner join should be performed to further reduce the result set.
- The rank function should then be performed to determine rank.
- Finally the result set should be prepared for presentation. The GROUP BY clause should be executed to eliminate extraneous results and execute the sum() function to determine final rank for the result this. This should be concluded by the ORDER BY clause to order the results by rank (and subsequently by date).

6. Evaluation of OR-DBMS

The OR-DBMS I am using for my project is PostgreSQL 8.1. This is an SQL standard compliant open source object-relational database management system that I began using at the start of the semester. It includes support for most of the SQL3 enhancements that I wanted to implement in

the database. There are a couple of aspects that are confusing, or have sparse documentation, but on the whole, the documentation and technical features are quite good.

I suppose a lot of the confusion I had stemmed from being generally unfamiliar database languages. One aspect of PostgreSQL that I am still unclear about is its handling of large unstructured objects and its data type bytea. I feel the documentation here is minimal and not legible for someone with my user experience. This is one area where it is difficult to find implementation examples either within the offical PostgrSQL manuals or on the wider Internet.

There are also some (acknowledged) limitations of the DBMS that are either being worked on or will probably continue to exist due to internal structural reasons that are beyond my sphere of knowledge. One of these comes within the inheritance feature. While attributes are inherited from parent to child, primary keys and foreign keys are not. I have found this to be a frustrating feature in several test databases I have implemented, and either necessitate the creation of a lot more tables or completely rethinking the data model. Another limitation I noticed is the implementation of multivalued attributes and composite attributes. Each are a bit messy when it comes to SELECT's, UPDATE's, and INSERT's. I found multivalued attributes, supported with arrays, a bit too frustrating to use.

Still, there some great aspects, such as user-defined types, user-created functions, inheritance, and a full-text indexing module. The CREATE statements I have included give an example of these. The documentation and tutorials about these is very clear and the features are relatively simple to use. Another aspect of the DBMS that I like is the automatic optimization, described in detail in the manuals, along with good tips on how to structure your statements.

7. References

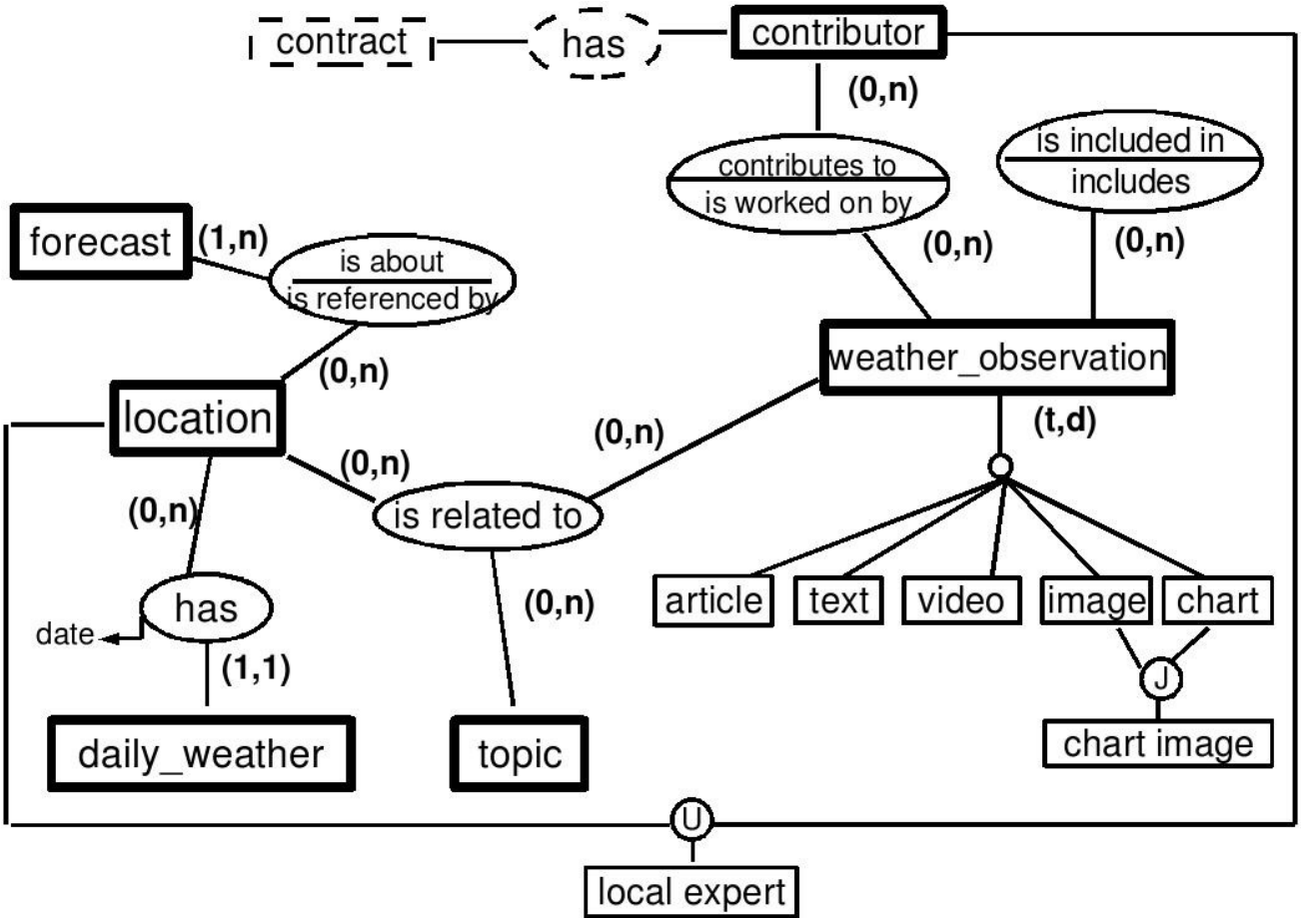
Lu, Guojun. **Multimedia Database Management Systems**. Artech House: Boston/London, 1999.

Nordbotten, J.C. **Multimedia Information Retrieval Systems.** 2006 <http://nordbotten.com/ADM/ADM_book>.

The PostgreSQL Global Development Group. **PostgreSQL 8.1.0 Documentation.** The PostgreSQL Global Development Group: 1996-2005.

8. Appendixes

8.a. Appendix A: Full Graphic Models



8.b. Appendix B: Attribute Lists

Multimedia Objects

|--**weather_observation**
- |--id <int>
- |--title <varchar>
- |--summary <varchar>
- |--format <varchar>
- |--date <date>
- |--text_index <tsvector>

|--**article**
- |--webpage <text>

|--**text**
- |--text <text>
- |--language <varchar>
- |--ft_index <tsvector>

|--**video**
- |--video <bytea>

|--**image**
- |--image <bytea>

|--**chart**
- |--(1,n) column_name <varchar[]>
- |--(1,n) data_array <float[]>

|--**chart_image**
- |--id
- |--type <varchar>

Other Entities

|--**topic**
- |--name <varchar>
- |--description <varchar>

|--**location**
- |--id <int>
- |--geo_loc <point>
- |--city <varchar>
- |--state <varchar>
- |--locex_id_pkey <int>

|--**daily_weather**
- |--id <int>
- |--min_temp <int>
- |--max_temp <int>
- |--precip <float>
- |--monthly_precip F<float>

|--**forecast**
- |--date_given <date>
- |--city <varchar>
- |--(1,n) date_fcast <date[]>
- |--(1,n) min_temp <int[]>
- |--(1,n) max_temp <int[]>
- |--(1,n) outlook <outlook[]>

|--**contributor**
    |--id <int>
    |--name <fullname>
        |--first <varchar>
        |--last <varchar>
    |--locex_id_pkey <int>

    |--**contract**
        |--id <int>
        |--date <date>
        |--salary <float>

|--**local_expert**
    |--id <int>

Relations

|--**isrelatedto_location**
    |--resource <int>
    |--location <int>
|--**isrelatedto_topic**
    |--resource <int>
    |--topic <varchar>
|--**includes**
    |--aid <int>
    |--mid <int>

8.c. Appendix C: Create Table Definitions

```
CREATE TABLE weather_observation
(
  id serial NOT NULL,
  title varchar NOT NULL,
  summary varchar,
  format varchar,
  date date,
  text_index tsvector,
  CONSTRAINT wo_pkey PRIMARY KEY (id)
);

CREATE TABLE text
(
```

```
  text text,
  language varchar,
  ftindex tsvector,
  CONSTRAINT text_pkey PRIMARY KEY (id)
) INHERITS (weather_observation);


CREATE TABLE includes
(
  aid int4 NOT NULL,
  mid int4 NOT NULL,
  CONSTRAINT inc_pkey PRIMARY KEY (aid, mid)
  CONSTRAINT inc_art_fkey FOREIGN KEY (aid)
     REFERENCES article (id) MATCH SIMPLE
     ON UPDATE RESTRICT ON DELETE RESTRICT
  CONSTRAINT inc_med_fkey FOREIGN KEY (mid)
     REFERENCES weather_observation (id) MATCH SIMPLE
     ON UPDATE RESTRICT ON DELETE RESTRICT
);


CREATE TYPE fullname AS
   ("first" varchar,
    "last" varchar);


CREATE TYPE results AS
  (id INTEGER,
   headline TEXT,
   rank REAL);
```

```
CREATE TRIGGER tsvectorupdate BEFORE UPDATE OR INSERT ON text
        FOR EACH ROW EXECUTE PROCEDURE tsearch2(ftindex, text);


CREATE FUNCTION findobjct(text) RETURNS SETOF results LANGUAGE sql AS '
  SELECT id, headline(text, q), rank(ftindex, q)
    FROM text, to_tsquery($1) AS q
    WHERE ftindex @@ q ORDER BY rank(ftindex, q) DESC';


CREATE OR REPLACE FUNCTION insert(varchar, varchar, text) RETURNS void LANGUAGE
sql AS
  'INSERT INTO text (title, summary, text, text_index)
  VALUES ($1, $2, $3, setweight(to_tsvector($1), "B") || setweight(to_tsvector($2), "C") ||
to_tsvector($3));';


CREATE INDEX ftindex_indx ON text USING gist(ftindex);
```