REMOTELY-SENSED GEOPHYSICAL DATABASES: EXPERIENCE AND IMPLICATIONS FOR GENERALIZED DBMS

Guy M. Lohman
IBM Research Laboratory
San Jose, CA 95193

Joseph C. Stoltzfus, Anita N. Benson, Michael D. Martin
Jet Propulsion Laboratory
Pasadena, CA 91109

Alfonso F. Cardenas
UCLA & Computomata International Corp.
Los Angeles, CA 90024

ABSTRACT

This paper presents the characteristics of scientific remotely-sensed databases that are relevant to -- and pose unique challenges for -- general-purpose database management systems (DBMSs). We describe a prototype system that integrates geophysical data and its metadata from both satellite and *in situ* sources, using a relational general-purpose DBMS to manage the catalog and observational data, and a video optical disk to archive images. Based upon our experience with this application, we suggest augmentations to DBMSs that would facilitate their use not only for scientific databases, but also for engineering, document, and even commercial database applications.

1. INTRODUCTION

General-purpose database management systems (DBMSs) have found wide applicability to commercial enterprises such as banking, inventory management, personnel information, etc. More recently, the application horizons of GDBMSs have widened to include geographical databases [MARB 77], [NAGY 79], [STON 80], pictorial databases (often for geographical applications) [BRYN 76], [BLSR 80], [COX 80], [CHAN 81], [CHOC 81], [ZBRS 81], engineering databases and CAD/CAM [LPTK 78], [BEET 82], [GUTT 82], [HASK 82a], [JOHN 82], [KATZ 82], and office documents [STON 82b]. Many of these novel applications have suggested enhancements to DBMSs that are required to better model those database types. This paper presents a similar analysis for scientific data that is collected by scientists using both spacecraft and ground *(in situ)* sources to study geophysical phenomena such as atmospheric and oceanographic processes.

* Portions of this research were carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract NAS 7-100 with the National Aeronautics and Space Administration, sponsored by the Information Systems Office, Office of Space Science and Applications.

Traditionally, DBMSs have not been used for these databases for a variety of reasons, some of which we will discuss below. Data from each spacecraft have been stored chronologically on magnetic tape using basic file access methods, usually in customized systems. Occasionally, DBMSs have been used to catalog and cross-reference the tapes, but integrity problems arise when the DBMS does not also manage the observational data on the tapes: there is no clear boundary between the "meta-data" in the catalog and the keys to the data on tapes. Users have increasingly requested greater integration of data from multiple sources, data independence, concurrency and integrity control for simultaneous access by multiple users, and more flexible access mechanisms via a high-level, general-purpose query language: all features provided by DBMSs.

In section 2, we first describe the aspects that characterize remotely-sensed geophysical databases and that pose unique challenges for DBMSs. Section 3 discusses the design and implementation of a prototype database that uses a relational DBMS to integrate oceanographic data and meta-data from satellite and *in situ* sources, as an enhancement to NASA's Pilot Ocean Data System (formerly called the Ocean Pilot System [BROW 82]). Section 4 suggests some augmentations to DBMSs that would enhance the implementation and performance of scientific remotely-sensed databases.

2. THE NATURE OF REMOTELY-SENSED GEOPHYSICAL DATA

In this section, we will characterize remotely-sensed geophysical databases, emphasizing those collected from spacecraft, the major source of our database. The most important of these aspects that pose unique challenges for DBMSs are: (1) the automatic, real-time collection of very large volumes of data via remote sensing instruments, (2) the regularity and structure with which a single spacecraft usually collects data, (3) the continuous domains and granularity of the data sampled, (4) the difficulty in relating scientific data from different sources, and (5) the stability of this data but dynamism of its structure.

2.1. Automatic Collection

Unmanned spacecraft are essentially robots with sensors aboard that are directed to measure some phenomenon such as the light reflected off a small portion of the Earth, and to radio that observation to a receiving station. Because this process is totally automated, data can be collected at far greater rates and in much larger volumes than in typical commercial applications where humans must capture and introduce data. Spacecraft data collection rates are limited only by the data rates of the sensors and the bandwidth of the radio link to Earth.

The sensors having the greatest data rates are imaging sensors, such as the Multi-Spectral Scanner (MSS) aboard Landsat 3 and the soon-to-be-launched Landsat D. See Figure 1. This type of sensor contains a row of photoelectric cells that convert light intensities in a narrow band of the electromagnetic spectrum into a vector of digital values called picture elements, or "pixels". Typically, the digital value is a byte (8 bits) that is capable of representing 256 distinct intensity values, but any number of bits may be used depending upon the intensity resolution desired. Complicated and very precise optics focus light onto the cells from a small portion of the Earth (about 60 meters square for the MSS). As the optics are moved mechanically or electrically, they sweep out a "footprint" on the Earth that is converted into a sequence of vectors, i.e. a two-dimensional array. This matrix of pixels is used on the ground to re-create a raster image of that footprint. All this time, the spacecraft is of course moving relative to the Earth, so the optics can sweep back to "observe" another footprint to be radioed in succession (see Figure 1).

A single image, and the ancillary information about that image that is appended to it during processing, requires so much storage that it must be stored in a series of files. As an example, the format for the MSS is shown in Figure 2 [HLKN 79]. The tape directory file contains attributes of the entire image, such as the spacecraft and time of observation. For each spectral band (color of light), there is a large image file surrounded by a scene attributes file and a trailer file. The scene attributes file has more detailed information about that image, such as how it was geometrically or radiometrically corrected, formatted, or otherwise processed. All but the image file are comprised of character, integer, real, or bit data types; each record of the image file typically contains a row of the picture that is stored as a record number followed by thousands of byte-sized pixels (see Figure 2). Note that to treat the entire image for one spectral band as a single entity would ideally require a 3240 by 2430 array of bytes.

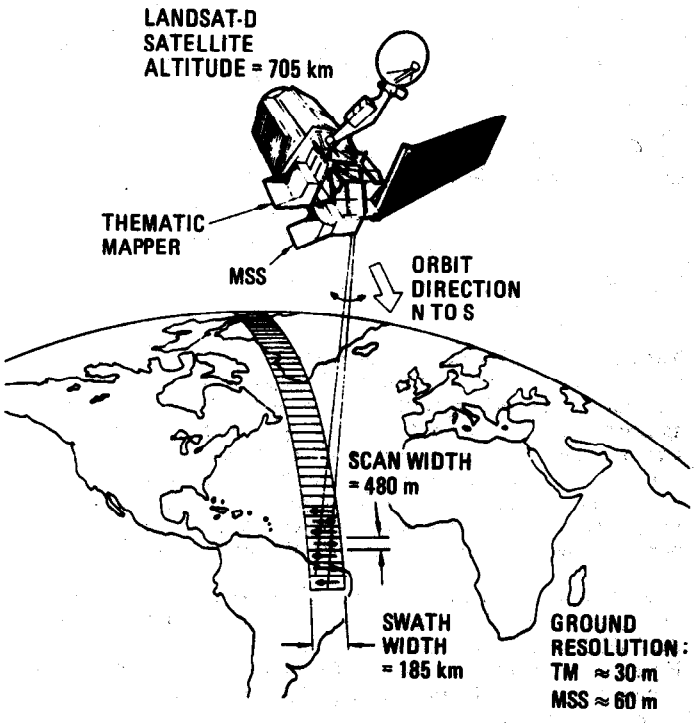


Figure 1: **Example of satellite remote sensing data acquisition (Landsat D [NASA 81]).**

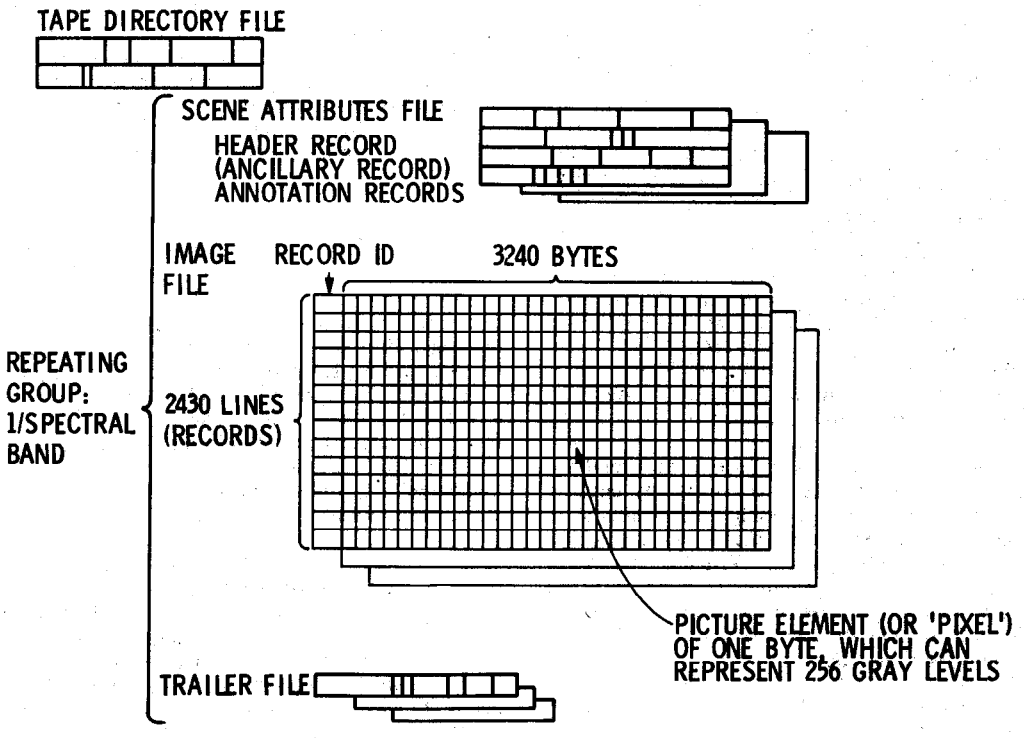


Figure 2: **Example format of satellite image files (Landsat 3 Multi-Spectral Scanner [HLKN 79]).**

The rates at which imaging sensors can generate bits is astounding. For example, the Thematic Mapper (TM), a 30-meter resolution imaging sensor also aboard Landsat D, will generate 85 megabits per second [ENGL 82]. Operating around the clock, imaging sensors are currently capable of generating 10^{11} to 10^{12} bits per year. Advances in sensor technology will make rates of 10^{15} bits per year likely by 1990 [BRCK 80]. The implications for database management are obvious and most challenging.

2.2. Regularity of Acquisition

The regularity with which automatic sensors can acquire data lends to the resulting database a natural structure that can be exploited for conceptually and physically organizing it. The trajectory of the spacecraft as a function of time is quite predictable, and the spacecraft attitude (orientation) and pointing angles of each sensor are either fixed or adjusted at known times by ground contollers. This makes the location of a sensor's footprint functionally dependent upon time in a calculable way. The calculation is geometrically non-trivial but accurate to a first level of approximation.

Most Earth-orbiting satellites are in either geosynchronous or low-altitude near-polar orbits. Geosynchronous satellites such as the GOES weather satellites and communication satellites orbit the Equator at altitudes high enough (25,000 miles) that their speed matches that of the Earth's rotation. Thus the satellite appears to be in a fixed position relative to an observer on Earth. Low-altitude near-polar satellites are more common for scientific observation, because their low altitude (typically about 700 km) permits finer spatial resolution and eventual observation of the entire Earth from the same viewing angle. The Landsat series of spacecraft, illustrated in Figure 1, have an orbit of this type.

Low-altitude orbits have a few disadvantages, however. Since only small portions of the Earth's surface are observed at any time, and since footprints of successive orbits ("swaths") are not adjacent, it is difficult and sometimes impossible to construct synoptic views of areas spanning more than one swath. For relatively stable phenomena such as the Earth's gravitational field, this presents no problem, but transient phenomena such as weather and ocean state change before the adjacent swath is available (sometimes days later). In addition, adjacent swaths may spatially overlap (particularly at the poles) or suffer gaps between them (especially near the Equator), as shown in Figure 3. Depending upon the location of stations available at any time for receiving the spacecraft's telemetry, similar gaps and overlaps in time can also occur. In addition, there is the usual problem in mapping geographical location to the database in a way that keeps adjacent locations close to one another in the database [NAGY 79]. However, this latter problem is not unique to satellite-acquired or even geographical databases, as it is simply the problem of wishing to cluster data in storage based upon multiple keys.

The major implication of satellite data collection for database management is that order of acquisition is a natural key because there is an obvious and predictable relationship that permits us to calculate the portion of the Earth's surface that is under observation at any time: the satellite's ephemeris. Visualize the swath viewed by the satellite's sensors as a continuous string, wrapped around a ball that is revolving in a direction at right angles to the direction of wrapping. Eventually the entire ball is covered by one or more layers of string. The function

$$\text{latitude, longitude} = f(\text{time}) \qquad (1)$$

is of course continuous and well-defined according to the laws of motion, so it can be mathematically inverted:

$$\text{time} = f^{-1}(\text{latitude, longitude}) \qquad (2)$$

where the time may not be unique if the spacecraft is orbiting. It is then easy to use time as the primary key for access, while exploiting equation (2) to calculate the time(s) a given spacecraft passed over a given latitude and longitude. We call this special case of functional dependency a "calculable functional dependency".

2.3. Continuous Domains

The domains from which science samples -- such as space, time, frequency, temperature -- are generally continuous. Commerce, by comparison, deals mostly with discrete objects such as people, departments, part numbers, types, dollars, etc. Representing continuous domains digitally poses the same problem for DBMSs that it does for other scientific programs: at what level of resolution should the domain be discretized? Put another way, at what granularity should we sample the domain? The finer the granularity, the more distinct possible values. For an index, for example, should the values 3.14159 and 3.1416 be treated as the same value? What about 3.14159 and 3.14160? What meaning does equality have for discretized values that were sampled from a continuous domain by different observers (see section 2.4)?

Similarly, we may wish to vary the granularity of display to see either detailed or summary (synoptic) phenomena [SHOS 82]. In other words, we need to zoom in or out for a more or less detailed view of the same data [STON 82c]. For example, to determine the best route from Los Angeles to New York, one observes a small-scale map that omits all but major highways. Such a map would be useless, however, to an observer trying to locate La Cienaga Blvd. in Los Angeles. The degree of detail must be determined by the user's application. Since scientists are interested in exploring both micro phenomena (e.g., wave formation in squalls, or coastal upwelling) as well as macro phenomena (e.g., major currents) using the same data, scientific remotely-sensed databases must provide a hierarchy of resolution. Coarser resolutions can be derived by aggregating the finer resolution values; however, the reverse is not possible. Scientists therefore often wish to retain all measured resolution. Not only does this increase the volume of observations, but also in the absence of standardization it makes it very difficult to compare data that has been collected at slightly different resolutions, as discussed below.

2.4. Differing Data Sources

The unique way in which individual scientists and state-of-the-art sensors collect data makes almost impossible the standardization that we take for granted in commercial applications. As a result, it is very difficult to relate two observations, even if they were collected at the same place at the same time. In such an environment, the importance of information about the data, or "metadata", is increased [McCA 82]. The meaning and derivation (or "pedigree") of individual fields in scientific databases must be much more thoroughly documented.

When an attribute called SALARY of entity EMPLOYEE is defined in a commercial database, we expect from experience that the item will express gross remuneration in dollars for an individual. The only question is whether it is in dollars per year, per month, or whatever. For the attribute WIND_SPEED, the meaning is not so well-defined, even to a meteorologist! Is it instantaneous (to what accuracy?) or averaged over 1 second, 1 minute, or 1 hour? Is it measured at the surface (what is the meaning of surface over the ocean -- Mean High Tide?) or at 10 meters or 19.5 meters or the height that an annemometer is mounted on a ship? With what kind of instrument was it measured (annemometer vs. wind balloon vs. microwave remote sensing)? Is it the raw observation or processed, and if processed using what

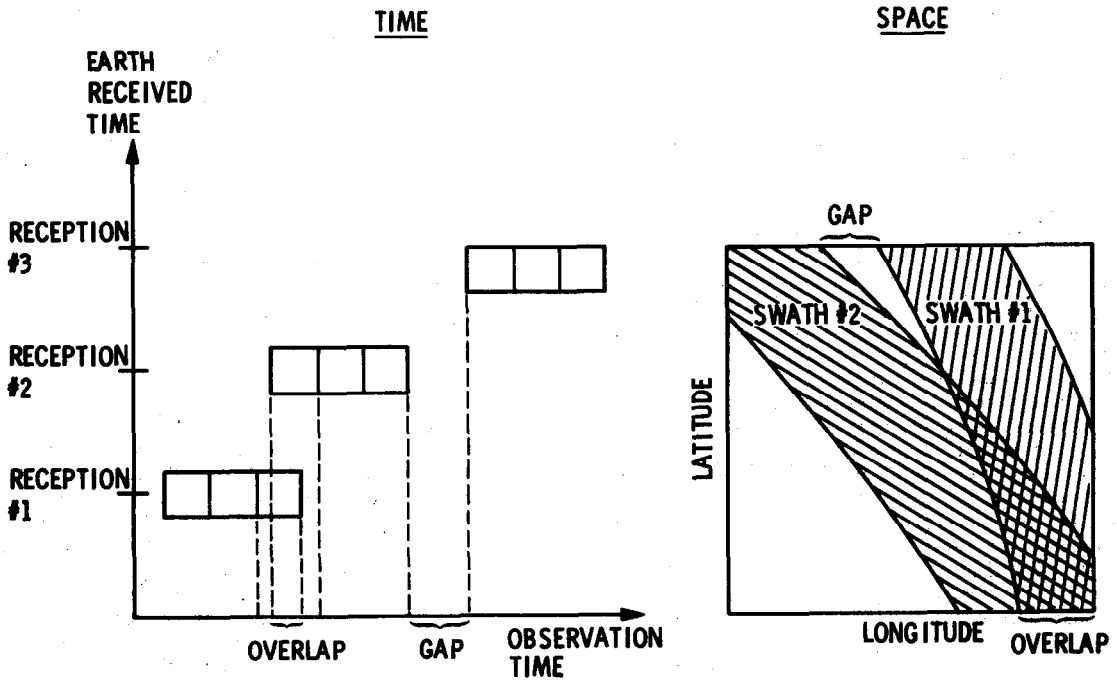


Figure 3: Gaps/overlaps in data from low-altitude satellites.

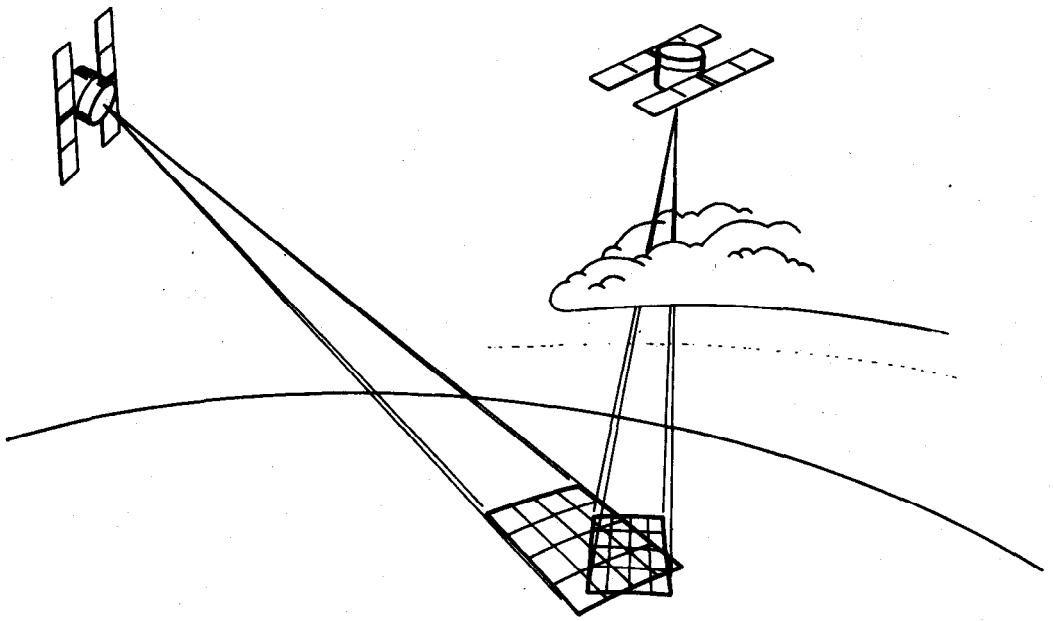


Figure 4: Differing observational geometries, footprints, resolutions, and atmospheric paths for 2 spacecraft viewing the same feature simultaneously.

algorithms, corrections, models, assumed constants? Does WIND_SPEED include the direction the wind is blowing, or is that WIND_VELOCITY? The role of the data dictionary in organizing this sort of information about the database must be greatly expanded to incorporate this vital "pedigree" or "catalog" or "meta-data" information that is so important to the scientist (see also [McCA 82], [SHOS 82]).

Even when so documented, scientific data from different sources may still be difficult to reconcile. As illustrated in Figure 4, images that are simultaneously taken of a common surface feature may nonetheless suffer from differences in observational geometry (viewing angle and spacecraft altitude), sensor resolution and "footprint" shape, and even different air columns through which the light traveled from the feature to the sensor. The latter aspect is important because moisture, particulates, and atmospheric thickness all affect the transmission of different frequencies of light in varying ways, acting as a filter or refractor. Scientists have devised image processing techniques such as rubber sheeting, resampling, and atmospheric modeling (respectively) to compensate for these differences, but the costs of these techniques make them prohibitive for all but a few objects of high interest: they could not, for example, be applied to all candidates of a query. Hence, even though data about a common feature on the surface of the Earth may have been observed simultaneously by two spacecraft and users wish to compare those observations, often that relationship is not exploited in a highly related database because the meaning of the comparison is questionable without at least knowing many other acquisition parameters and performing significant additional processing.

2.5. Static Data, Dynamic Structure

Researchers are loath to alter or discard observations that were acquired at great expense and whose value may increase with time as historical data. Therefore, scientific databases -- as statistical databases [SHOS 82] -- are altered over time almost exclusively by inserts rather than updates or deletes. This stability is advantageous in that it reduces the required concurrency control and recovery mechanisms, as well as the performance penalty for indexing many attributes [SHOS 82]. However, it poses significant data retirement and archiving problems. "Old scientific data never dies; it just gets 'archived' or reprocessed or restructured in new ways" (with apologies to Gen. MacArthur).

Often the relationships linking observed values cannot be completely defined *a priori,* and are very likely to change as analysis of the observations proceeds. In fact, determining the nature of such relationships might well define science. This dynamic structuring of observations argues strongly in favor of using relational DBMSs.

However, fairly static relationships exist between any observation and six selection criteria that appear to be universally used by scientists for database retrieval:

(1) Parameter (e.g., temperature or salinity)
(2) Methodology (i.e., the instrument, measurement techniques, and processing to acquire the value recorded)
(3) Place (i.e., geographical location)
(4) Time
(5) Quality (an unquantifiable amalgamation of assumptions, individual responsible, instrument calibration and malfunctions, precautions taken, uncertainties, degree of "cleaning" done, etc.)
(6) Groups or ranges of measured values [SHOS 82] (e.g., categories of values such as categories of wind-speed observations: "fresh breeze", "gale", "whole gale", "hurricane", etc.).

These latter, more static, relationships were the primary focus of our database design, discussed below.

3. PROTOTYPE DESIGN AND IMPLEMENTATION

A prototype database was implemented at JPL using data from the Joint Air Sea INteraction (JASIN) Experiment, which included measurements of meteorological and oceanographic parameters in a small part of the Atlantic Ocean near Great Britain during a 3-month period in 1978. Data from both *in situ* and satellite sources was simultaneously and intensively collected for the same region, and documentation of measurement methods and computer processing was readily available.

3.1. System Overview

A combination of DBMS software and custom application software with video display and storage hardware was designed to integrate image and non-image data. A simplified diagram of the system hardware and software is shown in Figure 5. The database and the DBMS are installed on a DEC VAX 11/780 running under the VMS operating system. A DEC VT100 terminal is the standard user device for query input and data display. A Sony video disk unit and a Sony video display monitor for storing and displaying image data have been tested and can be connected to the VAX by a modem and standard RS-232 cables. Images are stored in video (analog) format on the video disk unit; optical disks that store digitally formatted data are not yet commercially available [SBS 80].

The INGRES relational DBMS [RTI 82] using the QUEL query language provides access to the database. The INGRES Terminal Monitor is the standard interactive user interface to QUEL. The monitor allows entering, editing, saving, and executing queries, and provides an interface to standard VMS system services. It is possible to leave the INGRES monitor to execute another program and then return to INGRES with the context intact. A macro facility permits addition of user-written extensions to the query language, from abbreviations (e.g., DEFINE; ALG; algorithm;) to lengthy instructions with variable parameters defined at execution time. Application programs are written in FORTRAN with embedded QUEL statements. The QUEL statements can be tested interactively before they are included in the application program.

The interface to the video disk and display is still under development. The video disk unit displays a particular image on the video display in response to a character string input. The disk unit is designed to accept data in a standard terminal protocol, and has a limited capability for returning status data (found/not found, current track number) to the computer. A device handler program will be written to manage communication with the video disk unit. An application program whose query is satisfied by images in the database will receive the appropriate image numbers from the DBMS, which correspond to tracks on the video disk, and send them to the device handler program.

3.2. Database Design

As discussed in section 2.5, science data is typically retrieved using parameter, methodology, time, place, and/or range of measured values as primary selection keys. An example query (in QUEL) containing specifications of four of the keys (and abbreviating OBSERVATIONS as OBS) is:

```
RETRIEVE (OBS.ALL)
WHERE OBS.DAY#       = 78183
AND   OBS.PARAMETER = "*TEMPERATURE"
AND   OBS.SENSOR    = 1023
AND   OBS.LAT       >= 58.
AND   OBS.LON       < 63.
```

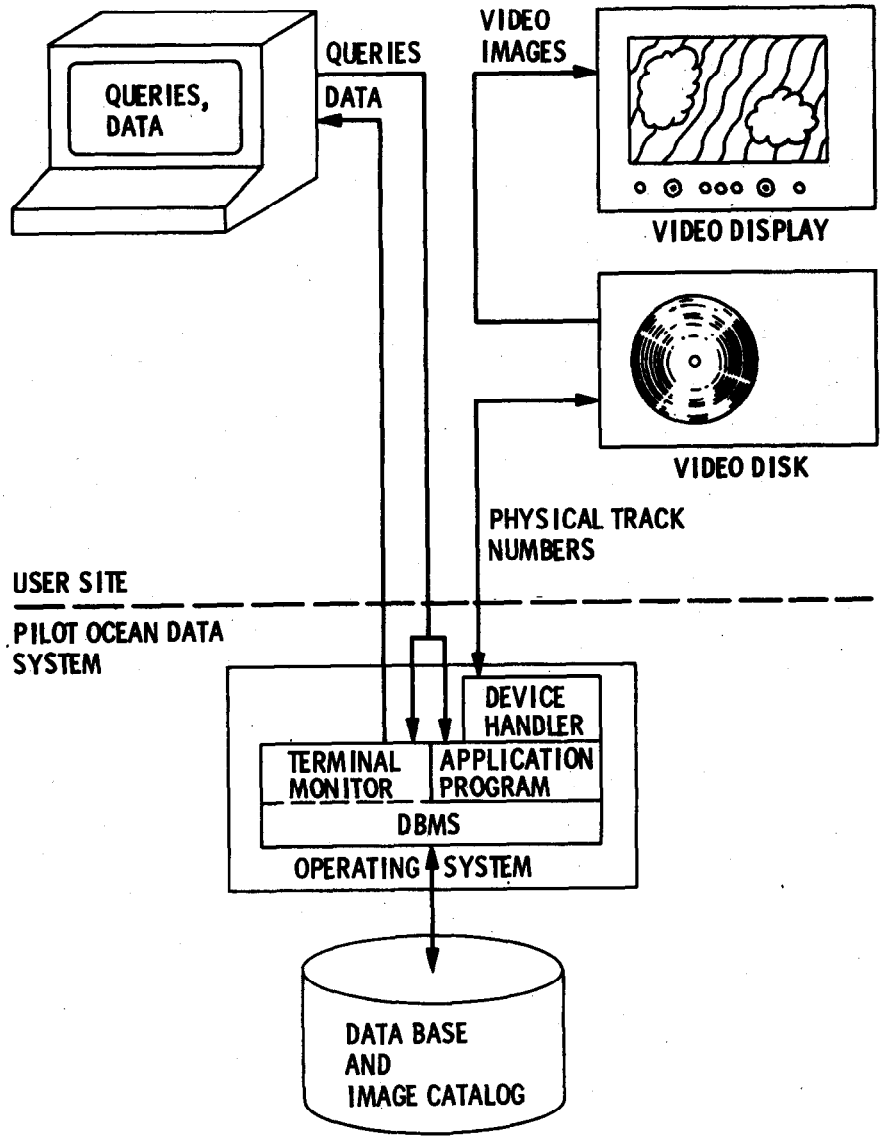


Figure 5: **Overview of prototype system configuration.**

Much effort was expended in developing a proper design for the database, in order to facilitate both user understanding of the data and efficient access and retrieval. For this we used the Entity-Link-Key-Attribute (ELKA) model [RAMY 83], an extension of Chen's Entity-Relationship model [CHEN 76]. The model proved to be a very useful communications medium between users (primarily oceanographers) and the system designers (database, image processing, and data cataloging experts). After many iterations, the design converged to that shown in Figure 6.

To aid the reader in understanding the design, let us review briefly the ELKA database modeling conventions. Each entity in the database is represented by a box in the ELKA diagram, with the name of the entity in the small box at lower left. Attributes whose values describe the entity are named in the box. Key attributes sufficient to uniquely identify each instance of the entity are underlined in the diagram. For example, in Figure 6, each instance of the entity named SENSORS has a unique identifier (SENSORS.SENSOR) and attributes describing that instance's type (SENSOR.SENSOR_TYPE) and platform (or vehicle) on which it is mounted (SENSOR.PLAT_NAME). The lines connecting the boxes are links representing one-to-many relationships between the connected entities, with the diamonds indicating the "many" end of those relationships. Open diamonds indicate that "many" may be zero, and solid diamonds indicate that "many" is at least one. Each instance of a "many" entity displays the key attribute(s) of the "one" entity to which it is related. The name of the relationship is written alongside the link. For example, in Figure 6, each PLATFORM instance CARRIES zero or more SENSORS instances, and each SENSORS instance is carried by a single PLATFORM. However, a named REGIONS instance must be DEFINED BY at least one instance within the REGION_DEFINITIONS table.

The major selection criteria listed earlier can be recognized as clusters of entities in the design:

The five entities at the upper left of Figure 6 describe the parameter and methodology, including sensor type, the particular sensor, the platform (e.g., spacecraft, ship, or airplane) which carried the sensor, and the parameter measured by that sensor.

The four entities at the lower left document the data quality parameters such as the principal investigator(s) responsible for collection and processing of the data, including the algorithms used and the degree (LEVEL) to which processing has progressed.

A single entity (OBSERVING_DAYS) was used to represent time because meteorological data was most commonly requested by integer days. The entity has attributes of both (1) year concatenated with (Julian) day number and (2) the triple (year,month,day) because both systems were in common use.

The three entities on the upper right permit describing geographical location either by name or by ranges of latitude and longitude. The JASIN experiment included five overlapping named areas, only one of which was rectangular and one of which was circular. It was decided to describe arbitrary regions in terms of 1-degree points (intersections of 1-degree lines of latitude and longitude) that are within each named region, effectively "binning" the data in 1-degree squares. The REGION_DEFINITIONS entity contains instances of these points.

The OBSERVATIONS (or OBS) entity includes the actual measurements, identified uniquely by time, place, sensor, parameter, and processing history. Since inclusion of all possible parameters in a single table would result in many null values when only a few parameters were measured, we elected to include only a single measured parameter per OBSERVATION. This high degree of normalization leads to much redundancy of key values for each measurement (see also [SHOS 82]). The DATA_SETS entity was introduced to remove some of the redundancy in the OBSERVATIONS entity by combining day, sensor, and parameter into a single short parameter, effectively forming a hierarchy (see section 4.3 below for a discussion of our need for hierarchies).

The IMAGES entity, a yet-to-be-implemented entity for storing images in digital form, is related to the same entities in the same way as OBSERVATIONS, but has different attributes such as SUN_ELEVATION that are unique to images (see sections 4.1 and 4.3 below for a discussion of ways to handle digital image data types and their treatment as a special case of OBSERVATIONS).

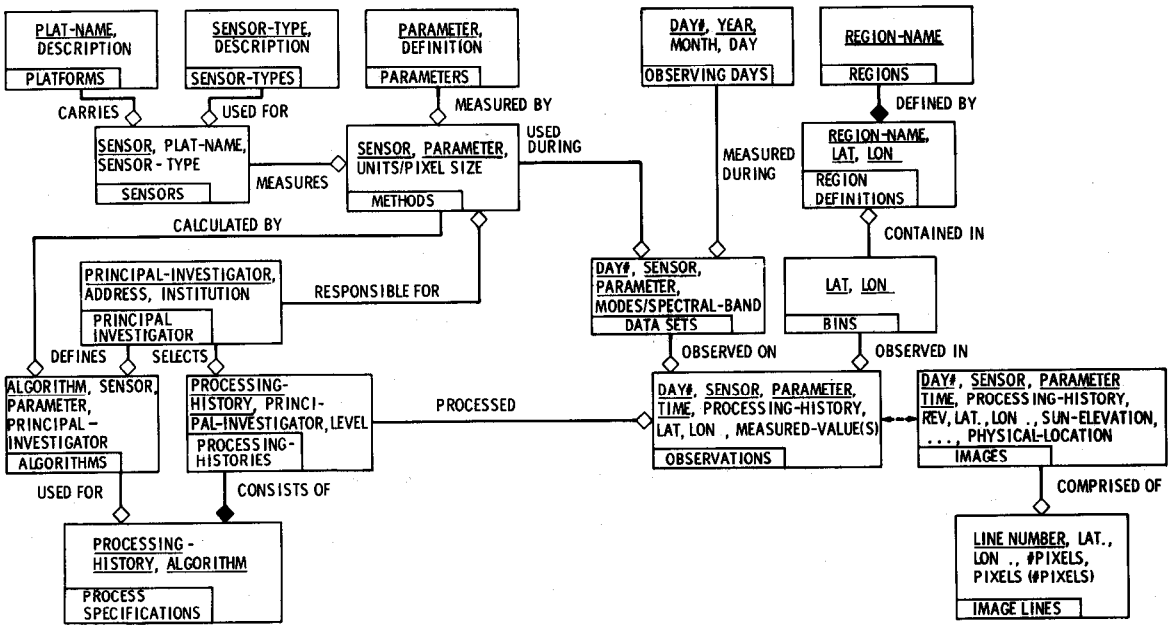


Figure 6: **Map of *in situ* and image catalog data base, using ELKA model [RAMY 83].**

The reader should verify that the database design supports the following typical queries that may be posed by different specialists:

- *sensor scientist:*
  Retrieve all data measured by an anemometer on the ship John Murray.
- *discipline scientist:*
  Retrieve all wind vectors for the Meteorological Triangle region.
- *principal investigator:*
  Retrieve all data collected by Paulson.
- *pedigree:*
  Retrieve the names of all principal investigators whose algorithms have processed wind data.
- *image analyst:*
  Retrieve all images in the JASIN area from July 22.

3.3. Implementation Experience

The prototype database was implemented directly from the ELKA model design, with each entity represented by a relational table. *In situ* data from the JASIN experiment were loaded and tested first because they are richer in structure and more manageable in quantity than the satellite data. However, the database design applies equally to both remotely-sensed and *in situ* data.

The OBSERVATIONS table is the largest in the database, and will become far larger (>100,000 records) when all the JASIN data are loaded. The input data consisted of groups of observations from different sources, with records containing from one to ten parameters measured at a single time and place. Re-structuring these into our normalized OBSERVATIONS table made the loading process slow and complicated. Determining which instrument measured a given parameter required manual comparison of text files, file names, published documents, and labeled drawings. Determining the processing history of the observations in terms of which algorithms had been applied required even more manual detective work. These are typical problems in gathering scientific data for a database.

Once loaded, retrieval of data by joining even three tables proved to be limited primarily by CPU and/or page swaps. Performing complicated multi-table joins in a step-by-step manner, i.e., as a series of two-table joins, proved easier for a number of reasons. Two-table joins were faster, preserved joins for later use, and provided intermediate results to the user that made a complicated join easier to understand and that confirmed correct progress. However, the intermediate results required storing additional tables that were both redundant and unnormalized, leaving the database vulnerable to inconsistency and update, insert, and delete anomalies.

Table proliferation was a problem when results were saved in physical tables to avoid the delay of repeated RETRIEVEs. After some weeks, the database became laced with forgotten tables, which the users were reluctant to purge without examination but which required too much time to examine. In a production environment, this could not be tolerated. The use of views would save the physical storage space, but at the expense of repeating expensive joins whenever the view was used.

The definition of regions in terms of 1-degree points is convenient and easy to use, even for odd-shaped regions such as ocean coastlines. Although each region that is defined requires a detailed list of points that it contains, this need be done only once to define a region, and combining multiple regions into a single region only requires appending tables. Since the same region is typically used repeatedly and rarely -- if ever -- changes, defining new regions is required infrequently. Selection of geographic data by region requires a join, for example (in QUEL):

```
RETRIEVE ( OBS.ALL )
WHERE OBS.LAT = REGION_DEFINITIONS.LAT
AND   OBS.LON = REGION_DEFINITIONS.LON
AND   REGION_DEFINITIONS.REGION_NAME
            = "HYDROGRAPHIC SURVEY AREA"
```

As discussed above, for larger regions, performance is improved by creating a new table by saying:

```
RETRIEVE INTO HSA ( REGION_DEFINITIONS.ALL )
WHERE REGION_DEFINITIONS
            = "HYDROGRAPHIC SURVEY AREA"
```

and thus simplifying the geographic selection to:

```
RETRIEVE ( OBS.ALL )
WHERE OBS.LAT = HSA.LAT
AND   OBS.LON = HSA.LON
```

3.4. Future Plans

The two major extensions planned are (1) the loading of satellite data and (2) the integration of the image catalog and display.

A large number of satellite observations are available from the Pilot Ocean Data System for the same time period and regions as the JASIN *in situ* data currently in our database. Loading the satellite data is a straightforward process, but was postponed due to disk space restrictions on the prototype. We anticipate no fundamental changes to the database design, and achieving the important scientific work of intercomparing concurrent satellite and *in situ* data of the same meteorological phenomena.

To demonstrate the video disk, 10 images from the Advanced Very High Resolution Radiometer (AVHRR) imager aboard Tiros-N and approximately 400 images from the Synthetic Aperture Radar (SAR) aboard Seasat, some of which coincide in time and region with the JASIN experiment, have been recorded on an analog laser video disk, at an approximate cost of $0.70 per image. Random access to the 50,000 images storable on one side of the video disk, which may be controlled like a terminal, has been demonstrated under the control of an APPLE microcomputer. A stub program has demonstrated access to the database and control of any terminal-like device by an application program invoked from the Terminal Monitor of INGRES. It remains only to link these two portions by developing a device control program for the optical disk on the VAX, so that it may be invoked by our application program, and to enter the appropriate image catalog data for each image on the video disk.

4. DBMS REQUIREMENTS FOR REMOTELY-SENSED DATA

The implementation of our application would have been simplified, and its power and performance considerably enhanced, with a DBMS having the following capabilities. A more comprehensive but less detailed discussion of requirements for this type of database can be found in [LOHM 81]. Many of the requirements of [McCA 82] for management of metadata are also relevant to our application, because much of our database could be considered to be metadata. In fact, a major motivation for our database was the recognition that the boundary between data and its meta-data is so fuzzy as to be undefinable.

We discuss below additional requirements upon DBMSs that are implied by remotely-sensed data, and briefly introduce some possible solutions. The syntax of suggested commands is given as variations of SQL only to lend some precision to the examples, in a form familiar to most readers. Prototypes of some of these capabilities are under development, but to our knowledge no commercially available system has them all.

4.1. Data Types

Additional data types have been suggested by numerous authors for various reasons. Many of these are needed for remotely sensed data bases as well. Any data base containing scientific observations will need REAL (or FLOAT) and BINARY (or HEX) data types. Some combinations of bits that are generated by scientific instruments are not valid characters, and thus must be manipulated as binary values. Although many commmercially available systems do support REAL, they seldom support the BINARY data type. REAL and BINARY numbers stored as TEXT strings are inadequate because they cannot be used in arithmetic expressions such as (in SQL):

```
SELECT MEASURED_VALUE, GAIN
FROM   OBSERVATIONS
WHERE  MEASURED_VALUE**2
                BETWEEN 6.2E3 AND 7.9E8
AND    GAIN BETWEEN 'A0'X AND 'FF'X;
```

Furthermore, the storage of one bit as a byte when storing it as a character is unacceptable for large volumes of data.

ARRAY data types are required for any grouping of homogeneous data elements, such as time series observations and raster images [CHOC 81, SHOS 82]. Unstructured fields of arbitrary length (not limited to 32K bytes as LONG VARCHAR fields are in SQL/DS) are needed to store large objects such as images that may be treated by the DBMS as a single, unstructured object at some times [HASK 82a]. For example, when browsing the catalog for an item of interest, the user retrieves an entire image, graphics file, or document into his own personal data base. There, however, his processing applications require structuring of the object into arrays so that he can access an individual pixel, curve, or line of text. For example, catalog entries would be created (in SQL) as:

```
CREATE TABLE IMAGES(
   DAY#            INTEGER,
   TIME            INTEGER,
   SENSOR          INTEGER,
   PHYSICAL_LOCATION CHAR(12),
   IMAGLEN         INTEGER,
   IMAGE           OBJECT(IMAGLEN) );
```

where PHYSICAL_LOCATION contains the physical device and IMAGLEN contains the length in bytes of the object called IMAGE. The user would MIGRATE (see [HASK 82b]) the tuple containing the amorphous object IMAGE into a table declared in his database space:

```
CREATE TABLE MYIMAGS(
   DAY#            INTEGER,
   TIME            INTEGER,
   SENSOR          INTEGER,
   PIXELS (2430,3240)  BINARY(8) );
```

so that queries can place conditions on individual pixels in a predicate without having to transfer the entire object to the application program:

```
SELECT PIXELS(I,J)
FROM   MYIMAGS
WHERE( PIXELS(I-1,J-1) + PIXELS(I-1,J) +
       PIXELS(I-1,J+1) + PIXELS(I  ,J-1) +
       PIXELS(I  ,J) + PIXELS(I  ,J+1) +
       PIXELS(I+1,J-1) + PIXELS(I+1,J) +
       PIXELS(I+1,J+1)  ) / 9
  BETWEEN 'A0'X AND 'FF'X;
```

This query retrieves all pixels whose intensities, when averaged with all neighboring pixels, are between 'A0' hex and 'FF' hex.

Extensible data types, popular in programming languages such as Pascal, would enhance the semantic meaning and integrity-checking capabilities of DBMSs independent of application programs, and would obviate the need for adding application-specific data types (such as the "geographic data type" suggested by [COX 80]) cluttering the syntax of the DBMS. Overmyer and Stonebraker [OVER 82] have implemented "time" data types using "experts", user-defined functions that are invoked from the DBMS to encode, decode, compare, and perform arithmetic operations on the application-specific data type using database tables to translate. They have also generalized experts somewhat to so-called "abstract data types", where one can "DEFINE ADT name" and procedures to convert, compare, and perform arithmetic operations on these types [STON 82b]. It is not clear how these mechanisms differ from experts. And unlike extensible data types, they do not appear to permit nesting of type definitions to build complex types. For example, one would like to construct an application-specific data type by combining elemental data types and predicates:

```
CREATE DATATYPE LATITUDE
    = INTEGER BETWEEN -90 AND +90;
CREATE DATATYPE LONGITUD
    = INTEGER BETWEEN -180 AND +180;
CREATE DATATYPE LOCATION
    = (LATITUDE,LONGITUD);
```

Not only would the database better reflect the application, it could also better match the host application language, and could facilitate the addition of new object types when using the DBMS to store meta-data [McCA 82].

4.2. Table Attributes

Tables in the original relational model had no explicit or implicit ordering of either its columns or rows (tuples), and no attributes other than the columns it contained and authorization information such as the owner and password. Only KEY columns received special treatment.

Stonebraker et al. [STON 82b], [STON 82c] have noted the need to order tuples in a relation, particularly those containing lines of text. In [STON 82b], they propose ordering a table with a command (in QUEL):

```
ORDER table-name BY field-1
        <WITH field-2 = ASCENDING field-3>
                        DESCENDING
```

where the optional portion in brackets is a secondary ordering. Field-1 becomes a line identifier that the user can see and manipulate in queries. It is indexed by a special B-tree having a line count in all but the leaf nodes, which contain tuple identifiers. This and other text manipulation commands proposed by [STON 82b] would prove useful in manipulating textual descriptions in our meta-data concerning data collection techniques, processing performed, underlying assumptions, caveats, etc. (not shown in our design).

However, ordering, like indexing, is often known when the table is created, and hence should be part of the schema for that table. We therefore propose that tables, like columns, can have attributes that differentiate how the system should treat them.

One such attribute would be ORDERED:

```
CREATE mydoc ( text=c132 ) ORDERED
        <BY (field-1,<field-2>,...)>
```

Omitting the BY clause would signal the system to create and maintain the ordering field in the relation automatically and totally transparent to the user.

A table could also have the attribute VERSIONED, in order to permit the user to retain complete and consistent earlier versions of the same large table (e.g., a design or document), only a small proportion of which changes at a time [HASK 82b]. VERSIONED tables would have the updates for each version stored in a differential file [SEVE 76] rather than updating the table in place, until a SIGNOFF command indicates the version to be an official, approved version. Until SIGNOFF, the last signed-off version remains read-only. At SIGNOFF time, all of the updates since the last SIGNOFF (or in designated differential files) would be applied to the last signed-off version, creating a new, "clean", official version to work from.

4.3. Hierarchies

Several authors have documented the need for -- and possible approaches to -- hierarchically structuring tables in other applications, particularly for CAD/CAM and statistical applications [HASK 82a], [HASK 82b], [KATZ 82], [McCA 82], [SHOS 82], [MEIE 83]. Hierarchies occur in several ways in our application:

4.3.1. Hierarchies of Resolution

Section 2.3 mentioned the hierarchy of increasing resolution, with data at each level representing the aggregation of the next lower level of the hierarchy. However, re-deriving the aggregations each time they are needed may be preferable to storing them in a hierarchy of resolution.

4.3.2. Partitioning Large Tables into Equivalence Classes

As noted in section 3.2 above, efficient implementation of hierarchies would also help reduce redundancy and permit partitioning of very large tables. For example, the OBSERVATIONS table in our prototype database is very large and has a good deal of redundant key values, notably DAY#, SENSOR, and PARAMETER (see Figure 6). If a hierarchy were to be constructed by making the OBSERVATIONS entity a child of the DATA_SETS entity, using the COMP_OF construct of [HASK 82b], then OBSERVATIONS would be partitioned into equivalence classes having a common DAY#, SENSOR, and PARAMETER, which need be specified only once in the parent DATA_SETS instance (see Figure 7). While this reduces the redundancy within OBSERVATIONS, we still have not truly partitioned OBSERVATIONS, because a join with DATASETS will involve *all* instances of OBSERVATIONS unless we index both DATA_SETS.DID and OBSERVATIONS.DID and join them with a semi-join.

4.3.3. Different Tables with Common Columns

Hierarchies also arise in our application because tables having slightly different schemas may have columns of interest in common. For example, in our prototype database, the OBSERVATIONS and IMAGES tables differ only by a few columns, such as SUN_ELEVATION. To retrieve the sensor and parameter information for all IMAGES *or* OBSERVATIONS (in fact, for all tables having those columns) in the database that were acquired on a particular date, we wish to say (in SQL):

```
SELECT *.SENSOR, *.PARAMETER
FROM   *
WHERE  *.DAY# = 78183;
```

Currently, the only alternative to this query would be for the user to (1) submit a query to the data dictionary to find the names of tables having attributes SENSOR, PARAMETER, and DAY#; (2) write down the table names; and then (3) repeatedly submit the query, each time using a different table name in the same query. There is no way to do it from an application program, short of sneaky uses of SYNONYMs.

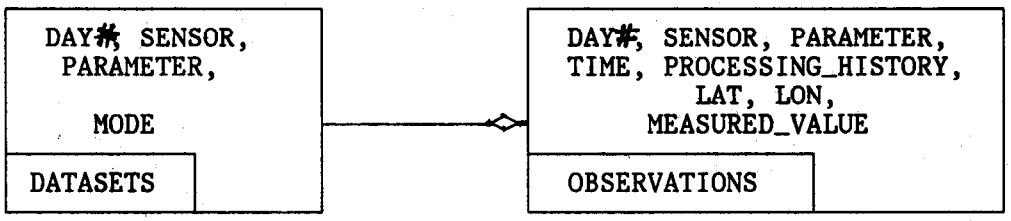
4.3.4. Table Names as Query Variables

The problems described in sections 4.3.2 and 4.3.3 can be solved by permitting table names in a query to be variables to be determined by the result of another, earlier do-at-open (i.e., uncorrelated) query. That result must be a single-column table (i.e., a vector) of valid names of tables that that user is authorized to access, which was either built as a temporary table by a nested query [SELI 79],[KIM 82], or is currently displayed on the screen. This differs from the "transitive closure" suggested by [GUTT 82] in that it is not limited to all progeny of a tuple in a single, fixed hierarchy. Rather, any table that can be the result of a query can provide the names of the table (see example below). If no predicate constrains the choice of a table name query variable (as in the example in Section 4.3.3), then all tables in the data dictionary having all the columns referenced in the query would be candidates. It is left up to the user as to whether his query makes any sense semantically; the system only checks at bind time that the table name is a valid one.

In most current systems that pre-compile queries, the table names cannot even be application program variables, because until they are known the system cannot perform catalog lookup, authorization, optimization, binding, etc. With table names being variables, pre-run-time compilation of the query would not be possible, a significant performance objection unless these extensions are used primarily for interactive queries. However, they provide a powerful and elegant extension to the query language that:

(1) Permits queries of the type given above;
(2) Permits partitioning very large tables into equivalence classes that will be accessed only if they satisfy a query that specifies those keys that define the equivalence class; and
(3) Saves the occasional user from having to remember table names in a large database.

ELKA Model of Hierarchy:



Haskin and Lorie schema [HASK 82b]:

DATASETS

DID	DAY#	SENSOR	PARAMETER	MODE
IDENTIFIER	INT	INT	CHAR(14)	CHAR(2)

OBSERVATIONS

OID	DID	TIME	PROCESSING _HISTORY	LAT	LON	MEASURED _VALUE
IDENTIFIER	COMP_OF(DATASETS)	INT	CHAR(12)	INT	INT	REAL

Lohman *et al.* schema:

DATASETS

DAY#	SENSOR	PARAMETER	MODE	OBS_TAB_NAME
INT	INT	CHAR(12)	CHAR(2)	CHAR(8)

OBS_TAB_NAME

TIME	PROCESSING _HISTORY	LAT	LON	MEASURED _VALUE
INT	CHAR(12)	INT	INT	REAL

Figure 7: **Two approaches to Hierarchies: IDENTIFIERS vs. Variable Table Names.**

As an example of the second advantage, suppose we added to the DATASETS table (see Figure 6) a column called OBS_TAB_NAME, which contained a table name for all OBSERVATIONS instances having the same DAY#, SENSOR, and PARAMETER. Those columns could then be eliminated from OBSERVATIONS (see Figure 7). Then the following query (in SQL) would access *only* those OBSERVATIONS with the specified DAY#, SENSOR, and PARAMETER:

```
SELECT MEASURED_VALUE
FROM   X
WHERE  X IN
   (SELECT OBS_TAB_NAME
    FROM   DATA_SETS
    WHERE  DAY#      = 78183
    AND    SENSOR    = 109
    AND    PARAMETER = 'WIND*')
AND TIME = 140000;
```

We have designed a true hierarchy in which parents point to children, whereas Haskin and Lorie effectively have children point to parents. We are therefore vulnerable to the update, delete, and insert anomalies of a true hierarchy, but as we noted in section 2.6, the database of this application is relatively static. Which approach is better will depend upon the application.

4.4. Storage Device Interfaces

DBMSs today assume that all the data they manage is on magnetic disk. Data on tape or mass store is assumed to be transferred to disk by the operating system before being accessed by the DBMS. However, in our design for the prototype, the catalog contains addresses of data that may be on media other than the traditional magnetic disk. And as the database grows, lesser-used data will have to be archived offline but retain its entries in the catalog. Hence the physical device type on which the data reside must be explicitly maintained by the DBMS, because access methods will vary depending upon the storage medium. For example, catalog entries for IMAGE entities in our database must note if the image is in analog form on the video disk, or in digital form on magnetic disk. If the former, our video disk device controller must be invoked and passed the desired track number; if the latter, the usual magnetic disk interfaces through INGRES to the operating system are invoked. However, such physical device considerations should be transparent to the user, as data migrates from one device to another.

DBMSs will in the future have to interface with an increasing variety of storage media types, between which data may migrate dynamically. While the operating system will have to provide access primitives for each type, the DBMS will still be responsible for formatting the data, providing concurrency control and recovery mechanisms, keeping track of what data is on what device (or else DBMS catalog entries will point to incorrect locations), and invoking the proper primitives depending upon device type. Even a single table may have its attributes partitioned among devices, so file level migration is inadequate. For example, our IMAGES table has the key attributes on magnetic disk but the corresponding (analog) image is stored on optical disk, linked by the device track number.

This argues for an expanded internal schema, or even another layer of schema that maps the internal schema (which presumably is device-independent) to a device-dependent schema [CODA 78], [LOOM 80]. Such a mechanism would also permit the database administrator greater flexibility in specifying system storage format and parameters such as page size, locking granularities, etc. to facilitate tuning performance. Although this would significantly change an existing storage subsystem such as System R's RSS, new systems could incorporate the multi-level schema concepts of IPIP [JOHN 82] that permit mapping between any number of schemas using explicit mapping schemas.

4.5. Application-Specific "Hooks"

As Stonebraker has pointed out, semantic extensions to DBMSs are needed to enable it to better model the application [STON 82a], yet not all applications want to be burdened with the complexity of the additional syntax. For example, we needed to define extensions of QUEL that would measure the distance between two points or determine if two regions intersect. While such functions would also be useful in CAD/CAM, not all applications would benefit by such an addition.

As mentioned in section 4.1 above, Overmyer and Stonebraker originally implemented "hooks" to user-specified functions, called "experts", to implement additional data types such as "time" [OVER 82]. However, the functions invoked could perform only one of four specific tasks: encoding a new data type into an internally coded value, decoding that internal value, comparison, and arithmetic on these data types. More recently, Stonebraker [STON 82a] has proposed collapsing the encoding and decoding function types into a single conversion function type. He also added a general-purpose user-defined function type that permits the user to define any external function that is invoked by its appearance in an expression in the query, much as can be done in NOMAD [NCSS 77].

While the conversion, comparison, and arithmetic function types permit a simple implementation of abstract data types, we feel that Stonebraker's syntax blurs unnecessarily the distinction between user-defined routines and the definition of user-specific data types. Furthermore, it is not clear whether user-defined functions can have abstract data types as parameters, as this would force nesting of routines. The Pascal-like syntax for extensible data types presented in section 4.1 would permit this nesting and would leave only one function type: the general-purpose, user-defined function.

An example in our application will illustrate the broad usefulness of user-defined functions with the syntax we propose. For example, if we have a FORTRAN subroutine called DISTANCE to calculate the distance between two points X and Y, then invoking that function directly from a query saves retrieving all tuples to the application program and invoking the function there:

```
/* a vector in 3-space is defined by its 3 coordinates. */

   DEFINE DATATYPE COORDINATE = INTEGER;
   DEFINE DATATYPE VECTOR = COORDINATE(3);

/* create a table full of points, each point being a
     3-tuple.  table POINTS contains attribute POINT.  */

   CREATE TABLE POINTS (POINT OF TYPE VECTOR);

/* define a distance function whose parameters are points
   having type "vector", i.e., are specified as 3-tuples. */

   DEFINE FUNCTION DISTANCE(VECTOR,VECTOR)
      ( <filename of FORTRAN subroutine> ,
          FORTRAN, <type returned>       );
      :
      :
/* use function called DISTANCE to retrieve
     all pairs of POINT instances in POINTS
     that are within 5.3 units of each other,
     using cursors T1 and T2 on table POINTS.  */

   SELECT T1.POINT, T2.POINT
   FROM   POINTS T1, POINTS T2
   WHERE  DISTANCE( T1.POINT,T2.POINT ) <= 5.3;
```

The DEFINE would bind the DBMS's internal name DISTANCE to the filename containing executable code for the function, and would reserve space for the parameter list, if any. The language in which the function is defined must be specified because of varying parameter-passing conventions among languages.

Hooks of this sort provide a powerful mechanism for tailoring the query language to a particular application. For example, using functions similar to but more complex than the DISTANCE function, we could expand the query language to include all basic spatial relationships between geometric objects -- adjacency/connectivity, containment, intersection/overlap, distance, and direction -- where the function invoked would be different for objects of different dimension (points vs. lines vs. areas) [TSUR 80].

4.6. Browsing Capabilities

Several published ideas for browsing data in other applications are practically a necessity in browsing remotely sensed data.

One such idea is binding a relation to a user-specified "window" on the screen, and allowing the user to browse the relation by moving the cursor up and down and zooming in or out [STON 82c]. A useful augmentation would be to allow the user to define another "left or right" cursor on the same or another relation, so that the user could see tuples keyed by two attributes (e.g., latitude and longitude) specified by movement of the mouse. Another enhancement would be to permit queries against only those tuples that are designated with TIMBER's PICK command, as can be done with the MARK command of NOMAD [NCSS 77], instead of -- or in addition to -- a predicate:

```
SELECT SPECTRAL_BAND, MEASURED_VALUE
FROM   PICKED OBS O, PICKED DATA_SETS D
WHERE  O.DAY# = D.DAY#
AND    TIME  < 140000;
```

The keyword PICKED qualifies the table name that follows it, to denote the subset of that table's tuples that were designated with the cursor while browsing.

To browse a large relation efficiently, the user should also be permitted to specify that he wishes to display only every nth tuple. Though this does not limit the data retrieved from disk, sometimes the user's terminal is the limiting resource (e.g., over a low-speed phone line).

Another idea useful to remotely-sensed databases is Haskin and Lorie's CHECKOUT/CHECKIN commands [HASK 82b], which perform concurrency control at a very macro level and are analogous to a librarian [KATZ 82]. As with Shoshani's statistical users [SHOS 82], our users typically browse the entire database interactively looking for data of interest, then transfer that subset to a personal database for intensive study and processing (which may update the retrieved data) over a period ranging from hours to months. The CHECKOUT and CHECKIN capability is required for these long-period transactions to prevent two researchers from altering the same data simultaneously or using the data unaware that another researcher is altering it simulaneously (but offline).

In browsing large remotely-sensed databases, the scientific user typically adds predicates until a manageable but not too small set of observations has been designated. Therefore, the first response to any query, *in browse mode only,* ought to be an estimate of how many tuples satisfy the query, so the user can judge whether the system should bother to retrieve them all or not [McCA 82]. In a system such as System R, the optimizer estimates the size of the result of a query for assessing costs, and could make that result available to the user before executing it. For queries whose predicates affect only indexed columns, the exact size of the result is calculable by accessing only the indexes and applying the predicates to their values, the result of which is a set of tuple identifiers (TIDs) satisfying the query.

To facilitate the narrowing of queries based upon the results of earlier queries, the user could define each query as a view and then pose his next query against the view, i.e. FROM <view-name>. However, this becomes awkward if repeated: either the views become nested, or the user has to re-enter the entire query for each new view. While Interactive SQL (ISQL) permits the labeling of queries for recall of the text, nesting of the label within another query is not permitted: the user can STORE the query, RECALL it later, and then edit it with the CHANGE command.

We propose a simpler mechanism whereby the system prompts the user for a new query with a system-assigned query number:

<query number>: <query>;

as is done, for example, in CCA's Model 204 [CCA 80]. The system automatically stores the query syntax for the last N queries. Then the query number can be used to substitute for the query text anywhere in another query. Thus, we could express (in SQL) our nested query in section 4.3.4 as:

```
1: SELECT OBS_TAB_NAME
   FROM   DATA_SETS
   WHERE  DAY#     = 78183
   AND    SENSOR   = 109
   AND    PARAMETER = 'WIND*'
```

to review the intermediate results (the table names), and then

```
2: SELECT MEASURED_VALUE
   FROM   X
   WHERE  X IN 1
   AND    TIME = 140000;
```

to see the desired data.

4.7. Encoded Values and Value Synonyms

Because relational systems establish relationships on values rather than through explicit pointers, redundancy of values can become troublesome. There is a trade-off between long values that the user understands and an encoded value that conserves space internally [SHOS 82]. Also, we sometimes wish to establish a synonym relationship between several values. For example, 'CALIFORNIA', 'CALIF', and 'CAL' -- all meaning the same thing to the user -- would be better represented internally as 'CA'.

An encoding and decoding mechanism more similar to the ENCODE and DECODE constructs of NOMAD [NCSS 77] than the conversion functions of Stonebraker would be very useful to most applications. This involves creating a conversion table having two columns, one for full-length values and one for encoded values, e.g.:

STATES

SNAME	CODE
CALIFORNIA	CA
CALIF	CA
CAL	CA
CA	CA
MICHIGAN	MI
MICH	MI
NO STATE	null
	null
none of the above	null

"ENCODE (table-1,value)" would scan table-1 for the *first* occurrence of "value", and return its matching code. Conversely, "DECODE (table-1,code)" would do the reverse. Thus, multiple values could have the same code (or vice versa), with the first one being the preferred conversion. A default "none of the above" can be named in case the value or code does not occur in the table. Thus, in the above example, a value of blank or "FRANCE" will be ENCODEd as null, and upon DECODE will read "NO STATE".

Any appropriate column name can replace the value or code value in queries:

```
SELECT REGION_NAME,
    DECODE(STATES,REGIONS.STATE_CODE)
FROM REGIONS;
```

would convert internally abbreviated state codes in the table REGIONS to state names, using the table STATES. Note that this is different than simply doing a join on the conversion table STATES. Clearly the functions ENCODE and DECODE could be implemented using functions (see section 4.5).

Such a table-driven encoding mechanism is easy for the user to define and maintain in table-oriented relational systems, and is easier for the user to understand than writing a special routine to perform this common task. More complex conversions could, however, be performed with a user-defined function.

5. CONCLUSIONS

Our prototype has demonstrated the feasibility of using a relational general-purpose DBMS for integrating remotely-sensed geophysical data and its meta-data in a database that includes digital data and images stored on video optical disks. Spacecraft as well as *in situ* data were combined into a unified database design. While the DBMS met all our expectations, our experience has suggested some enhancements to DBMSs that we feel will be applicable beyond our immediate application. For example, all applications have need for more semantically meaningful data types such as TIME, DATE, or NAME, and ARRAY data types are common in engineering and statistical databases. ORDERED tables are essential to document databases, VERSIONED tables to design databases. Hierarchies occur naturally in commercial applications, e.g. in organization charts, as well as in engineering and statistical ones. User-defined functions, browsing features, and value encoding/decoding add customizing power to any application. These applications have many requirements in common with those of remotely-sensed databases that we highlighted, and, together, represent a significant future market for DBMS technology with the enhancements that we have suggested.

6. ACKNOWLEDGEMENTS

Steve Pazan of JPL and the Scripps Institute of Oceanography, La Jolla, CA, provided invaluable insight into the selection and use of oceanographic data during the design phase of our prototype. We also wish to thank Chuck Klose, John A. Johnson, and Jim Brown of JPL's Pilot Ocean Data System for their encouragement, advice, helpful technical discussions, JASIN data and documentation, and machine resources throughout the development of the prototype. Finally, we gratefully acknowledge Al Ferrari of JPL for inspiring, and Peter Bracken and Tony Villasenor of NASA for funding, the prototype database development.

REFERENCES

[BEET 82] A. Beetem, J. Milton, and G. Wiederhold, "Performance of Database Management Systems in VLSI Design", *IEEE Database Engineering 5,* 2 (June 1982), pp. 15-20.

[BLSR 80] A. Blaser (ed.), *Data Base Techniques for Pictorial Applications,* Florence, Italy, 20-22 June 1979, Lecture Notes in Computer Science, Springer-Verlag, New York, 1980.

[BRCK 80] Peter A. Bracken, "Earth Observation Data Systems in the 1980's", *Proc. of 1980 Annual Mtg. of American Astronautical Society,* Boston, MA, 20-23 Oct. 1980.

[BROW 82] James W. Brown, "Controlling the Complexity of Menu Networks", *Comm. of ACM 25,* 7 (July 1982), pp. 412-418.

[BRYN 76] Nevin A. Bryant and Albert L. Zobrist, "IBIS: A Geographic Information System Based on Digital Image Processing and Image Raster Datatype", *Machine Processing of Remotely Sensed Data,* IEEE, 1976, pp. 1A-1 to 1A-7.

[CCA 80] Computer Corporation of America, *Model 204 User's Manual,* Cambridge, MA, 1980.

[CHAN 81] Shi Kuo Chang and Tosiyasu L. Kunii, "Pictorial Data-Base Systems", *Computer 14,* 11 (November 1981), pp. 13-21.

[CHEN 76] Peter Pin-Shan Chen, "The Entity-Relationship Model -- Toward a Unified View of Data", *ACM Transactions on Database Systems 1,* 1 (March 1976), pp. 9-36.

[CHOC 81] Margaret Chock, Alfonso F. Cardenas, and Allen Klinger, "Manipulating Data Structures in Pictorial Information Systems", *Computer 14,* 11 (November 1981), pp. 43-50.

[CODA 78] CODASYL Data Description Language Committee, "CODASYL Data Storage Description Language, 1978", *Journal of Development,* January 1978.

[COX 80] Nicholas J. Cox, Barry K. Aldred, and David W. Rhind, "A Relational Data Base System and a Proposal for a Geographic Data Type", *Geo-Processing 1,* (1980), Elsevier Scientific Publishing Co., Amsterdam, pp. 217-229.

[ENGL 82] J.L. Engel and Oscar Weinstein, "The Thematic Mapper: an Overview", *Proc. of International Geophysical Remote Sensing Symposium,* IEEE Geophysics Section, Munich, June 1982.

[GUTT 82] A. Guttman and M. Stonebraker, "Using a Relational Database Management System for Computer Aided Design Data", *IEEE Database Engineering 5,* 2 (June 1982), pp. 21-28.

[HASK 82a] R. Haskin and R. Lorie, "Using a Relational Database System for Circuit Design", *IEEE Database Engineering 5,* 2 (June 1982), pp. 10-14.

[HASK 82b] R. Haskin and R. Lorie, "On Extending the Functions of a Relational Database System", *Proceedings of the ACM SIGMOD 1982 International Conference on Management of Data,* (July 1982), pp. 207-212.

[HLKN 79] Patrick F. Holkenbrink, *Manual on Characteristics of Landsat Computer-Compatible Tapes Produced by the EROS Data Center Digital Image Processing System,* Version 1.0, Change 1, USGS/NASA, U.S. Govmt. Printing Office, No. 024-001-03116-7, 1 Aug. 1979.

[JOHN 82] H.R. Johnson and D.L. Bernhardt, "Engineering Data Management Activities within the IPAD Project", *IEEE Database Engineering 5,* 2 (June 1982), pp. 2-9.

[KATZ 82] R.H. Katz, "DAVID: Design Aids to VLSI using Integrated Databases", *IEEE Database Engineering 5,* 2 (June 1982), pp. 29-32.

[KIM 82] Won Kim, "On Optimizing an SQL-like Nested Query", *ACM Transactions on Database Systems 7,* 3 (September 1982), pp. 443-469. Also available as IBM Research Laboratory RJ3063, San Jose, Calif., February 1981.

[LOHM 81] Guy M. Lohman, "Generic Functional Requirements for a NASA General-Purpose Data Base Management System," JPL Internal Document No. 82-6, Jet Propulsion Laboratory, Pasadena, CA, 1 October 1981.

[LOOM 80] Mary E. S. Loomis, "The 78 CODASYL Database Model: A Comparison with Preceding Specifications", *Proceedings of the ACM-SIGMOD 1980 International Conference on Management of Data,* (May 1980), pp. 30-44.

[LPTK 78] Richard S. Lopatka and Thomas G. Johnson, "CAD/CAM Data Management Needs, Requirements and Options", *Engineering and Scientific Data Management,* NASA Conference Publication No. 2055, Proc. of Conf. at Langley Research Center, Hampton, VA, 18-19 May 1978, pp. 25-40.

[MARB 77] Duane F. Marble and Donna J. Peuquet, "Computer Software for Spatial Data Handling: Current Status and Future Development Needs", working paper, Geographic Information Systems Laboratory, State University of New York at Buffalo, 1977.

[McCA 82] J. L. McCarthy, "Metadata Management for Large Statistical Databases", *Proceedings of the Eighth International Conference on Very Large Data Bases,* Mexico City (September 1982), VLDB Endowment, Saratoga, CA, pp. 234-244.

[MEIE 82] Andreas Meier and Raymond A. Lorie, "Implicit Hierarchical Joins for Complex Objects", IBM Research Laboratory RJ3775, San Jose, Calif., February 1983.

[NCSS 77] National CSS, Inc., *NOMAD Reference Manual,* Norwalk, CT, June 1977.

[NAGY 79] George Nagy and Sharad G. Wagle, "Geographic Data Processing", *Computing Surveys 11,* 2 (June 1979), pp. 139-181.

[NASA 81] National Aeronautics and Space Administration, *Applications Notice for Participation in the Landsat-D Image Quality Analysis Program,* Washington, DC, 23 Oct. 1981.

[OVER 82] Ricky Overmyer and Michael Stonebraker, "Implementation of a Time Expert in a Data Base System", *SIGMOD Record 12,* 3 (April 1982), pp. 51-61.

[RAMY 83] Timothy L. Ramey, Robert R. Brown, Michael A. Melkanoff, and Guillermo Rodriguez-Ortiz, "The ELKA Information Model", *Information Systems* (to appear in 1983).

[RTI 82] Relational Technology, Inc. *INGRES Reference Manual,* Relational Technology, Inc., Berkeley, CA, 1982.

[SBS 80] SBS, Inc., *The Impact of Optical Disc Memories (Videodiscs) on the Computer and Image Processing Industries,* Strategic Business Services, Inc., San Jose, CA, May 1980.

[SELI 79] P. Griffiths Selinger, et al., "Access Path Selection in a Relational Database Management System", *Proceedings of ACM-SIGMOD,* May 1979. Also available as IBM Research Laboratory RJ2429, San Jose, Calif., August 1979.

[SEVE 76] Dennis G. Severance and Guy M. Lohman, "Differential Files: Their Application to the Maintenance of Large Databases", *ACM Transactions on Database Systems 1,* 3 (Sept. 1976), pp. 256-267.

[SHOS 82] Arie Shoshani, "Statistical Databases: Characteristics, Problems, and Some Solutions", *Proceedings of the Eighth International Conference on Very Large Data Bases,* Mexico City (September 1982), VLDB Endowment, Saratoga, CA, pp. 208-222.

[STON 80] Michael Stonebraker and Kenneth Keller, "Embedding Expert Knowledge and Hypothetical Data Bases into a Data Base System", *Proceedings of the ACM-SIGMOD 1980 International Conference on Management of Data,* (May 1980), pp. 58-66.

[STON 82a] Michael Stonebraker, "Application of Artificial Intelligence Techniques to Database Systems", Memorandum No. UCB/ERL M82/31, Electronics Research Laboratory, Univ. of CA, Berkeley, CA, 6 May 1982.

[STON 82b] Michael Stonebraker, et al. "Document Processing in a Relational Data Base System", Memorandum No. UCB/ERL M82/32, Electronics Research Laboratory, Univ. of CA, Berkeley, CA, 6 May 1982.

[STON 82c] Michael Stonebraker and Joseph Kalash, "TIMBER: A Sophisticated Relation Browser", *Proceedings of the Eighth International Conference on Very Large Data Bases,* Mexico City (September 1982), VLDB Endowment, Saratoga, CA, pp. 1-10.

[TSUR 80] Tateyuki Tsurutani, Yutaka Kasahara, and Masaru Naniwada, "ATLAS: A Geographic Database System--Data Structure and Language Design for Geographic Information", *Computer Graphics 14,* 3 (July 1980), pp. 71-77.

[ZBRS 81] Albert L. Zobrist and George Nagy, "Pictorial Information Processing of Landsat Data for Geographic Analysis", *Computer 14,* 11 (November 1981), pp. 34-41.