Data Vaults: Database Technology for Scientific File Repositories

Current data-management systems and analysis tools fail to meet scientists' data-intensive needs. A "data vault" approach lets researchers effectively and efficiently explore and analyze information.

Increased data volume from advanced observatory instruments and simulations has led to *data-intensive* scientific domains.[1] Scientific research requires efficient technology to manage and explore high-volume data repositories.

Scientists organize this data primarily in multitier, filed-based repositories. Metadata encoded in file names or managed by workflow systems let researchers navigate and search for data. Customized tools process and analyze data by blending data access, computational analysis, and visualization. Increased data volume leads to limitations, such as poor access and scalability, the inability to incorporate new studies, and difficulty deploying the customized software and scripting tools "near the data."[2]

Database management systems (DBMSs) approach these problems by processing information at the data storage site, providing flexible declarative queries that analyze and scale information to terabytes (TBs) of data. Applying a database system to a scientific application is difficult for several reasons: a state-of-the-art DBMS requires data to be loaded up front in the database, which is costly and tedious. A DBMS doesn't naturally understand and support external file formats specific to scientific domains, and it provides only limited processing capabilities for nonstandard data types. Also, it doesn't provide integrated access to existing external libraries or tools for analysis and visualization.

To address these problems, we created the *MonetDB data vault*—a virtual scientific data warehouse that lets scientists simply attach an external file repository to the DBMS using, for example, a Uniform Resource Identifier (URI), which efficiently and flexibly processes queries over relevant data. The data vault keeps the data in its original format and place, and simultaneously allows transparent metadata and data access and analysis using a query language. The data vault relieves scientists' pressure to migrate files, and provides extended functionality and flexibility. Moreover, high-level declarative query languages (such as SQL and SCIentific Query Language, or SciQL) let scientists experiment with novel algorithms.[3] Scientists can combine familiar external analysis tools with efficient in-database processing for complex operations, for which databases have shown to be beneficial. Transparent, just-in-time data loading reduces the "bootstrapping" costs of adopting a database solution for existing file repositories.

1521-9615/13/$31.00 © 2013 IEEE
Copublished by the IEEE CS and the AIP

Milena Ivanova
Netherlands eScience Center
Martin Kersten, Stefan Manegold, and Yağiz Kargin
Centrum Wiskunde and Informatica

DBMSs and Data-Management Architectures

Before we discuss the data vault's functionalities, let's consider some important background information. Software architectures for data-intensive

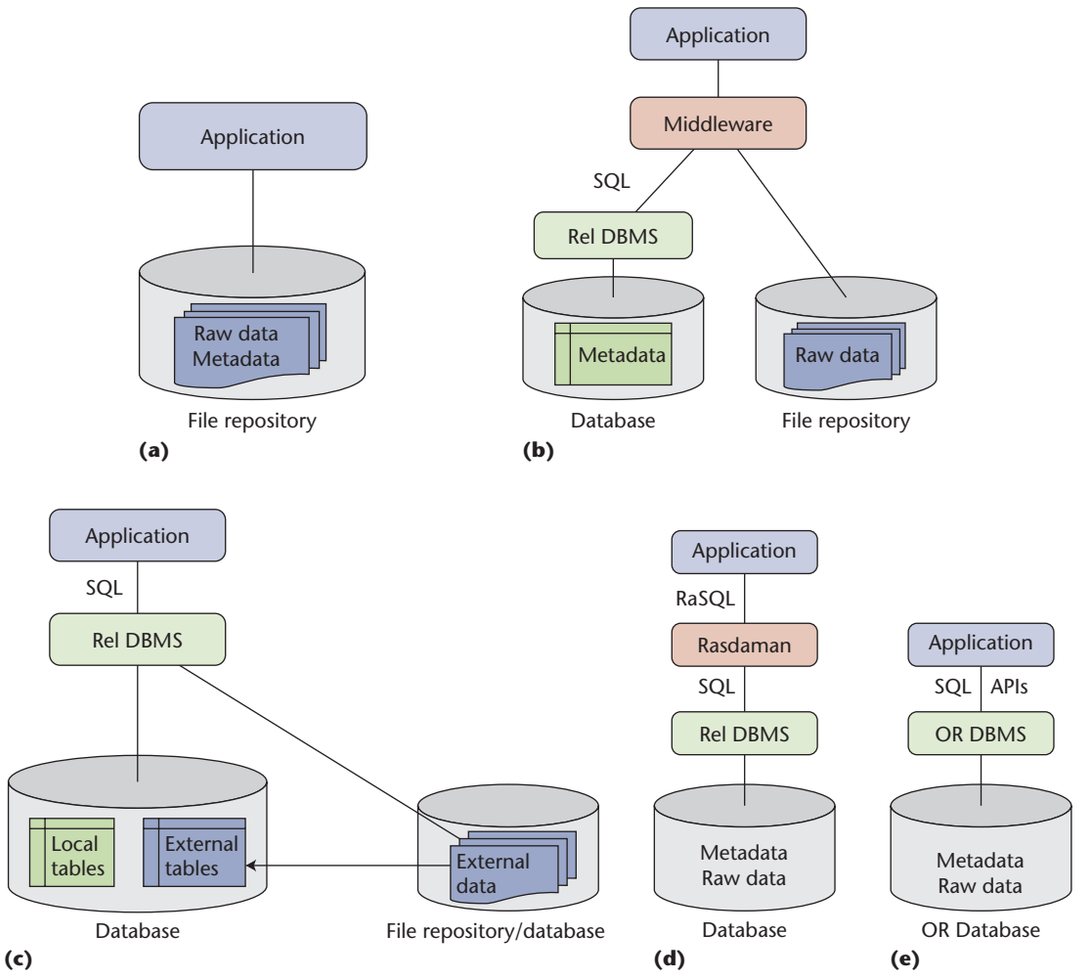

Figure 1. Software architectures for scientific repositories. Possible solutions include (a) file-based, (b) mixed solutions, (c) external tables, (d) Rasdaman, and (e) an object-relational (OR) database management system (DBMS). (Rel = relational).

scientific applications are diverse. Visual exploration tools, workflow systems, or large collections of programming scripts are often bound tightly together. Here, we focus on the most common data-management architectures and the role of DBMSs for them.

File-Based Solutions

Figure 1a illustrates one solution that uses a file-based repository for raw data and metadata. Most scientists take this approach, particularly in a small database's "long tail," for example, where they might use spreadsheets.

Mixed Solutions

As file-based repositories grew, mixed solutions became the larger, concerted scientific community's preferred tool to standardize file structures and manage metadata.[4] Custom-designed, mixed solutions often combine physical storage directives with file-system organization, and encode crucial information in the file names. Such solutions have important limitations, such as inflexibility when handling new requirements, inefficient access, and scalability.

Mixed solutions create inconsistency, and need different backup and access models, because they distribute file maintenance between the DBMS and the file system. Newer mixed solutions address this problem by providing some form of integration between DBMSs and file systems, mainly by facilitating file management and maintenance. Examples include the Oracle Database Filesystem (DBFS)[5] and the Microsoft SQL Server FileTable feature (http://msdn.microsoft.com/en-us/library/ff929144.aspx) that store files in the database and allow access to those files.

Although they can be used for any file type, the main application is for files with unstructured content (such as text and images) as needed by

content management systems. These systems typically limit file processing to a rudimentary, file-system-like interface that can upload and retrieve a file, update metadata, and manage directory structures. This category of systems doesn't aim to process file content in-database, but rather to efficiently provision files to external applications, such as text processing and content management systems.

Temporary DBMS Access to External Data

The SQL/MED (Management of External Data) standard proposes the external table feature that lets users access non-SQL (external) data as if it were in the database (see Figure 1c). The feature is applied predominantly to external data from other SQL-server vendors or in plain, comma-separated values (CSV) files.

The Universal File Interface extension (UFI; www.barrodale.com/universal-file-interface-ufi) offers an alternative for researchers indexing and querying external files. In addition to CSV files, it offers UFI adapters for scientific formats, such as network Common Data Form (NetCDF), Hierarchical Data Format 5 (HDF5), and Flexible Image Transport System (FITS; http://heasarc.nasa.gov/docs/heasarc/fits.html). Users must explicitly map the files and their elements to virtual tables, which they can then query from inside the database. Because users must specify the data they load and provide the mapping rules to the system, it is the user's task to localize and select data from the external file repository.

Raw Data inside DBMS

Figures 1d and 1e present architectures in which the DBMS stores and retrieves raw, nontabular science data. These systems chop the data into pieces and store it in Binary Large Object Block (BLOB) table columns. RasDaMan provides middleware with database services on multidimensional data structures, such as declarative query language, indexing, and optimization.[7] A DBMS back end stores and retrieves array tiles and indexes in BLOB form.

Modern object-relational systems offer various extensions that store, index, query, and retrieve complex data and their associated metadata. Examples are geospatial extenders for vector and raster data, such as Oracle Spatial and Graph (www.oracle.com) and PostGIS (www.postgis.org). The extenders come with various subprograms—user-defined functions (UDFs)—that offer common processing tasks over the new data types.

In both scenarios, the user must first explicitly ingest data into the system before the user can submit a query. Query processing on the middle layer or algorithms encapsulated in UDFs aren't (completely) integrated with the database internals, and the database optimizer might not be aware of it.

SciDB is a DBMS for scalable processing of array data.[8] Arrays are the only data model that SciDB supports, and query processing and optimization take advantage of their superior semantics and properties. Besides CSV files, the system offers file loaders for popular scientific formats, such as HDF5 and FITS. Like the previous examples, for all file types that SciDB supports, the user must first load the data before SciDB can process it. Currently, SciDB limits metadata support and management to the array structure. The system can't represent accompanying tabular metadata, such as sensor settings or experimental parameters encoded in the scientific formats. To do that, the user could artificially represent the metadata, for example, as a single-dimensional array that mimics a key-value list. Consequently, the system doesn't offer an integrated view over array data and tabular metadata, which would allow metadata-driven query processing.

NoSQL Solutions

Using the Hadoop and MapReduce coding style, programmers have been able to parallel process large file repositories. This might be the best option for users addressing parallel problems such as preprocessing, and aggregation over entire datasets; however, these coding styles require substantial resources. The MongoDB and Cloudera Distribution Including Hadoop (CDH) products, provide increasingly good declarative query languages and evolve toward a database system. Conversely, all database vendors include the MapReduce functional style in their products.

The main characteristics distinguishing the data vault approach from other architectures are its dynamic, on-demand data ingestion, support of array-based scientific formats, and integrated view over data and metadata that enables metadata-driven query processing.

To fully exploit the data vault features, the data vault should be a part of an array-enabled DBMS, such as RasDaMan, SciDB, or object-relational extenders for complex data types. The MonetDB multiparadigm data model treats both tables and arrays as first-class citizens. This special treatment provides adequate support to

array-based scientific formats, such as FITS, TIFF (http://trac.osgeo.org/geotiff/), and the Mini-Standard for the Exchange of Earthquake Data (mSEED).[9]

Systems that process array-structured raw data require users to statically and explicitly load data prior to query processing. Users must separately and often manually localize and select the raw data they want to load into the database. Associative search queries over the metadata drive the data vault to dynamically ingest external data.

Data Vault Architecture

Figure 2 illustrates the MonetDB data vault architecture. It extends MonetDB's software architecture with three components. The *data vault wrapper* communicates with the external file repository and accesses data, metadata, and external libraries and tools. The *data vault cache* manager takes care of the virtual data warehouse structure. The *data vault optimizer* searches for the best query execution plans over external data and drives dynamic data ingestion.

First, a user simply attaches an external repository specifying its location. Upon attachment, the system opens the repository metadata for browsing and sophisticated searching. Typical scientific formats encode a variety of metadata that enable applications to interpret the file contents. Correspondingly, scientific libraries usually offer lightweight functions that extract this metadata without touching the high-volume raw data. The data vault *metadata wrapper* component accesses the metadata and populates a predefined *data vault catalog*. This lets users quickly identify relevant datasets through declarative queries over the catalog. The catalog structure varies depending on the external data format. The data vault applies a lazy load approach to derived metadata that must be computed over the raw datasets. The application can piggyback the derivation on the first query over the data, or it can exploit idle CPU time.

Without further preparation and up-front data ingestion, the user can start submitting queries involving external data. For this purpose, the data vault supplies a virtual database structure that represents external data without actually loading them. We will load the data in a dynamic, just-in-time manner as needed by the queries. The data wrapper component accesses external data and creates internal representations in the data vault cache.

The data vault uses a symbiotic query-processing scheme. It loads queried data into the database and processes that data with pure database techniques. The data vault also uses external tools to process files in situ and to capitalize upon the existing support libraries. Symbiotic query processing combines the benefits of both approaches: use external tools, if efficient ones exist, and carry out operations in the database when the DBMS can perform them better. The data vault optimizer component provides this symbiotic query processing. Using the data vault catalog, it detects operations over repository data and makes decisions about their execution locations based on a cost model. If external tools process the query, the functionality wrapper component defines mappings between the external functions' input parameters and results to valid database representations.

If the optimizer deems in-database processing more efficient, it applies a two-phase optimization, at compile-time and at runtime, that facilitates automatic, on-demand data loading from files.[10] At compile time, the optimizer reorganizes the plan so that the query execution engine first evaluates the selection predicates on the metadata, which dynamically determines the actual files it will load. After the query execution engine performs this part of the plan, the optimizer triggers a rewriting operator at runtime. This operator modifies the rest of

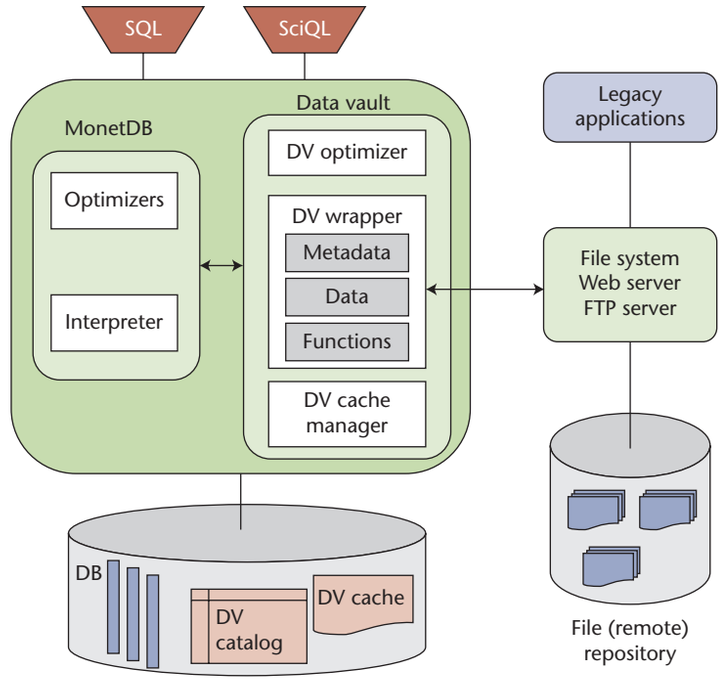

Figure 2. MonetDB data vault architecture. Three components extend the software: the data vault wrapper, the data vault cache manager, and the data vault optimizer.

the plan by replacing all references to external data with operators (data wrapper calls) that load the necessary files, or parts of them. Data wrapper operators then call to external scientific libraries to extract the data from the specific file formats.

The data vault lets file-based applications run as usual, because it keeps external repositories in their original format and place. Thus, it offers more functionality to application developers by extending existing library tools with database facilities, such as declarative queries, optimization, and efficient join operations.

If the external applications modify the file repository content, the data vault cache manager synchronizes the modification with the in-database representation, and thus always presents the user with an up-to-date repository state.

The data vault cache contains snippets of the repository data imported into the database as a side effect of query processing. This improves performance during subsequent analyses over the same external data. This component leverages our work on recycling intermediates.[11] The cache might contain the input dataset entirely or partially depending on the queries, cache size, and dataset size. Either the user or the system can apply diverse cache replacement strategies.[11] For instance, we can augment the cache with mechanisms that spill content onto the disk to accommodate large datasets. We can also transform the cached data into persistent database structures.

The data vault offers important features:

- It provides an integrated view over metadata and data—for instance, the data vault saves the user from having to manually browse the repository to determine which files might contain the dataset of interest; instead, the user can specify metadata selection criteria as an integrated part of the query, transparently translated into file-access instructions.
- It reduces the bootstrapping costs by reducing the time between external source data availability and the query answer.
- It enables declarative processing over external data, which is more convenient and efficient for users engaging in exploratory tasks and experimenting with new hypotheses and algorithms.
- It provides an alternative way for users to integrate heterogeneous data sources—for example, image processing can include images stored in both FITS and GeoTIFF formats.
- It gradually ports external archives into a DBMS solution that transforms the virtual data warehouse content into persistent database structures.

Use Cases

Here, we consider concrete examples of how we can use the MonetDB data vault's features in several scientific fields.

Seismology

The Earth's surface moves constantly, generating huge amounts of data. Scientists prefer the SEED file format to exchange waveform data among seismograph networks.[12] A SEED volume has several American Standard Code for Information Interchange (ASCII) control headers and highly compressed data records—that is, the waveform time series. The control headers store the metadata, which consists of identification and configuration information about the data records.

We applied our data vault to a collection of mSEED files extracted from the Observatories and Research Facilities for European Seismology (Orfeus; www.orfeus-eu.org) file repository, containing more than 3.5 million files collected since 1988. An mSEED file contains limited metadata and multiple mSEED records (35 records per file on average in our data collection). An mSEED record contains the sensor readings over consecutive time intervals, which in our data collection is usually a time series of approximately 3,500 values on average.

Data warehouse schema. We straightforwardly derive the normalized data warehouse schema shown in Figure 3 from the mSEED file format. The `files` and `records` tables hold the metadata per mSEED file and record, respectively, while the `data` table stores the actual sensor data. Each row in the file table identifies an mSEED file via its URI (file location), and contains the metadata describing the sensor that collected the data (`network, station, location, channel`) as well as some technical data characteristics (`dataquality, encoding, byte_order`). A unique sequence number identifies each record, and that record holds metadata such as the `start_time`, sampling rate (`frequency`), and number of data samples (`sample_count`). The `data` table stores the time series data as (`timestamp, value`) pairs. For user convenience, we define a (nonmaterialized) view `dataview` that joins all three tables into a (denormalized) universal table. Upon initializing the seismic data vault, only the metadata tables are populated, while the `data` table is empty. We use

```
CREATE SCHEMA mseed;

CREATE TABLE mseed.files (
  file_location STRING,        dataquality  CHAR(1),
  network       VARCHAR(10),   station      VARCHAR(10),
  location      VARCHAR(10),   channel      VARCHAR(10),
  encoding      TINYINT,       byte_order   BOOLEAN,
  PRIMARY KEY (file_location)
);

CREATE TABLE mseed.records (
  file_location STRING,        seq_no       INTEGER,
  record_length INTEGER,       start_time   TIMESTAMP,
  frequency     DOUBLE,        sample_count BIGINT,
  sample_type   CHAR(1),
  PRIMARY KEY (file_location, seq_no),
  FOREIGN KEY (file_location)
    REFERENCES mseed.files(file_location)
);

CREATE TABLE mseed.data (
  file_location STRING,        seq_no       INTEGER,
  sample_time   TIMESTAMP,     sample_value INTEGER,
  FOREIGN KEY (file_location)
    REFERENCES mseed.files(file_location),
  FOREIGN KEY (file_location, seq_no)
    REFERENCES mseed.records(file_location, seq_no)
);

CREATE VIEW mseed.dataview AS
SELECT
  f.file_location, dataquality, network, station, location, channel, encoding,
  byte_order, r.seq_no, record_length, start_time, frequency, sample_count,
  sample_type, sample_time, sample_value
FROM mseed.files AS f
  JOIN mseed.records AS r ON f.file_location = r.file_location
  JOIN mseed.data AS d
    ON r.file_location = d.file_location AND r.seq_no = d.seq_no;
```

Figure 3. Seismic data vault schema. Seismologists widely use the Standard for the Exchange of Earthquake Data (SEED) file format to exchange waveform data among seismograph networks.

the `libmseed` library (www.iris.edu/software/libraries) to extract (meta)data from mSEED files.

Query processing. Query processing over the seismic data vault varies from retrieving an entire record's data for visualization and visual analysis to aggregating and detecting outliers. The sample Query 1 in Figure 4 finds the maximum values (amplitudes) for a given channel (BHN, which is a broadband channel representing motion on the north–south direction) per station in the Netherlands (NL) during the first week of June 2010. Query 2 computes the long-term average (LTA, typically over an interval of 15 seconds) over the data generated at Kandilli Observatory in Istanbul (ISK) over all available channels.

The queries highlight the integrated view over data and metadata offered by the data vault. Predicates over the metadata select the data to be aggregated. At runtime, we modify the query execution plan to do the rest of the work. As we discussed previously, MonetDB seamlessly integrates the process of external data extraction and ingestion with query evaluation.

We performed experiments with three different datasets of increasing size. We randomly selected

```
-- Query 1
SELECT station, MAX(sample_value)
FROM mseed.dataview
WHERE network = 'NL' AND channel = 'BHN'
  AND start_time > '2010-06-01T00:00:00.000'
  AND start_time < '2010-06-07T23:59:59.999'
GROUP BY station;

-- Query 2
SELECT channel, AVG(sample_value)
FROM mseed.dataview
WHERE station = 'ISK'
  AND sample_time > '2010-01-12T22:15:00.000'
  AND sample_time < '2010-01-12T22:15:15.000'
GROUP BY channel;
```

Figure 4. Seismic queries. The sample Query 1 finds the maximum values (amplitudes) for a given channel (BHN, which is a broadband channel representing motion on the north–south direction) per station in the Netherlands (NL) during the first week of June 2010. Query 2 computes the long-term average (*LTA*, typically over an interval of 15 seconds) over the data generated at Kandilli Observatory in Istanbul (ISK) over all available channels.

5,000, 10,000, and 20,000 files, respectively, from a 2010 collection that contained more than 160,000 files. Table 1 lists some of the dataset's characteristics. The timestamp-value decompression of all samples causes the files to expand in size. This expansion characterizes the full, up-front file loading to MonetDB. Selective, just-in-time file loading as needed by the queries touches only a small portion of the raw data. This provides fast bootstrapping to the first query results. More comprehensive research examining the seismic data vault can be found elsewhere.[10]

Remote Sensing

We further explore the data vault architecture in the context of a remote-sensing repository to highlight MonetDB's usefulness when scientists design and experiment with new algorithms.

The example application comes from content-based image retrieval over remote-sensing images in the TELEIOS project (www.earthobservatory.eu). The source data are high-resolution TerraSAR-X (so named because of its active-phased array X-band synthetic-aperture radar, or SAR, antenna) satellite images in GeoTIFF format accompanied by metadata specifications in XML format. First, the example application prepares for the process by tiling an image into smaller chunks, called patches, and applying various feature-extraction methods over the patches. Customized software tools process image files in file repositories.

The application extracts features and stores them in a database, which it uses as input for higher-level image analysis, such as classification with support vector machines and content-based image retrieval. The data vault creates an opportunity to move the DBMS upstream in the processing pipeline, which provides flexible (and efficient) patch construction and feature extraction inside the database engine.

Data warehouse schema. The data warehouse schema (see Figure 5) contains two catalog tables with metadata and one array structure. The `files` table describes the GeoTIFF files in the repository with their location, status (loaded or not), and timestamp of the last modification.

The `image_catalog` table describes image-specific metadata. Here, we take an application-specific approach. Thus, in addition to the metadata encoded in the GeoTIFF files (image length and width), the table also contains auxiliary data extracted from the accompanying XML files (`sensor`), and properties encoded in the image file names (`resvariant`, `mode`, and so on). When the user attaches the image file repository, the metadata wrapper component browses the directory, extracts the metadata encoded in different sources, and populates the respective catalog tables.

Images map naturally to 2D arrays, therefore we chose to implement the remote-sensing data vault using the array-enabled SciQL language.[3] In addition to tables, the language lets users and applications create and directly operate over array structures. The user defines the arrays similarly to tables with one or more

Table 1. Datasets and their sizes.

Tuples per table			Dataset size		
Files	Records	Data	mSEED	MonetDB (full)	Data vault up-front load (metadata)
5,000	175,765	660,259,608	1.3 Gbytes	13 Gbytes	10 Mbytes
10,000	359,735	1,323,307,090	2.7 Gbytes	26 Gbytes	20 Mbytes
20,000	715,738	2,629,496,058	5.5 Gbytes	50 Gbytes	36 Mbytes

** mSEED = Mini-Standard for the Exchange of Earthquake Data.*

```
CREATE SCHEMA rs;

CREATE TABLE rs.files (
  fileid      INT,           location     STRING,
  status      TINYINT,       lastmodified TIMESTAMP );

CREATE TABLE rs.image_catalog (
  imageid     INT,           fileid       INT,         imagewidth INT,
  imagelength INT,           resvariant   CHAR(4),     mode       CHAR(2),
  starttime   TIMESTAMP,     stoptime     TIMESTAMP,
  sensor      VARCHAR(20),   absorbit     INT,
  PRIMARY KEY (imageid),
  FOREIGN KEY (fileid) REFERENCES rs.files(fileid) );

DECLARE NumCols INT;
SET NumCols = ( SELECT max(imagewidth)  FROM rs.image_catalog );
DECLARE NumRows INT;
SET NumRows = ( SELECT max(imagelength) FROM rs.image_catalog );

CREATE ARRAY images (
  id INT DIMENSION,             x INT DIMENSION [NumCols],
  y  INT DIMENSION [NumRows],   v SMALLINT );
```

Figure 5. Remote sensing schema. The `files` table describes the GeoTIFF files in the repository with their location, status, and timestamp of the last modification; the `image_catalog` table describes image-specific metadata; and the `images` 3D array represents the actual image data.

dimensional attributes. Those attributes are tagged with the `dimension` constraint, which essentially indexes the array tuples.

The `images` 3D array (see Figure 5) represents the actual image data. The first dimension is the associated image `id` and the other two are the image width `x` and length `y`. Again, the array is empty when the repository attaches, and the user doesn't ingest images up-front into the system. Instead, the data vault presents the virtual data warehouse to the users in the form of the images array so that they can formulate queries over images of interest. The `libgeotiff` library (http://trac.osgeo.org/geotiff) extracts (meta) data from GeoTIFF files.

Query processing. The first example in Figure 6 shows another integrated view over data and metadata—a table and an array. The SciQL query computes image masks by filtering pixel values within the range [10,100]. Predicates over the remote-sensing metadata specify the images—for example, the image resolution variant is spatially enhanced (SE), the imaging mode is high-resolution spotlight (HS), and the start time is in a given time interval.

The data vault optimizer recognizes the reference to the external repository in the form of the `images` array and rewrites the execution plan to ensure that the data wrapper ingests the images selected by the predicates over the `id` array attribute.

```
-- Query 1
SELECT [id], [x], [y], v
FROM images
WHERE v BETWEEN 10 AND 100 AND id IN
  (SELECT imageid
   FROM rs.image_catalog
   WHERE resvariant = 'SE__' AND mode = 'HS'
     AND starttime > TIMESTAMP '2011-12-08 16:30:00');

-- Query 2 Smoothing
SELECT [x], [y], avg(v)
FROM images[1][*][*]
GROUP BY images[1][x-1:x+2][y-1:y+2];

-- Query 3 Sampling
SELECT [x/4], [y/4], v
FROM images[1][*:4:*][*:4:*];
```

Figure 6. Remote sensing queries. Predicates over the remote-sensing metadata specify the images—for example, the image resolution variant is spatially enhanced (SE), imaging mode is high-resolution spotlight (HS), and the start time is in a given time interval.

Queries 2 and 3 demonstrate that users can easily specify typical operations, such as image

```
CREATE SCHEMA fits;

CREATE TABLE fits.files (
  fileid   INT,          location     STRING,
  status   TINYINT,      lastmodified TIMESTAMP );

CREATE TABLE fits.tables (
  tableid  INT,          name         CHAR(256),     columns INT,
  fileid   INT,          hdu          INT,
  PRIMARY KEY (tableid),
  FOREIGN KEY (fileid) REFERENCES fits.files(fileid) );

CREATE TABLE fits.columns (
  columnid INT,          name         VARCHAR(80),   type    VARCHAR(80),
  units    VARCHAR(10),  number       INT,           tableid INT,
  PRIMARY KEY (columnid),
  FOREIGH KEY (tableid) REFERENCES fits.tables(tableid) );

CREATE TABLE fits.images(
  imageid  INT,          fileid       INT,
  width    INT,          length       INT,
  PRIMARY KEY (imageid),
  FOREIGH KEY (fileid) REFERENCES fits.files(fileid) );
```

Figure 7. The Flexible Image Transport System (FITS; http://heasarc.nasa.gov/docs/heasarc/fits.htmlFITS) data vault schema. Because scientists can represent a whole database with various tables in a single file, we need a general structure that resembles a traditional SQL system catalog, instead of defining a fixed structure similar to the records table in the seismology data vault.

smoothing and sampling using declarative SciQL queries. The smoothing query uses a SciQL feature called *structural grouping*, which uses each valid value of the dimensions `x` and `y` to split the image into overlapping 3×3 pixel tiles. The pixel value at position ([x], [y]) in the result array is substituted with the average value for the corresponding tile centered at ([x], [y]).

The declarative interface lets researchers easily and conveniently experiment, and therefore boosts productivity.

Astronomy

Recent astronomy surveys often store their catalog data using database technology, but many archives use FITS files. With the growing number and scope of surveys, astronomers increasingly must correlate observations from different surveys over the same sky objects. Thus, the data vault might be valuable as a platform providing an integrated view over heterogeneous data sources.

Data warehouse schema. Astronomers widely use the FITS format to store and exchange images and tabular data. A FITS file can contain several header-data units that can be images or tables in ASCII or binary format. Scientists can represent a whole database with various tables in a single file. Consequently, instead of defining a fixed structure similar to the records table in the seismology data vault, we need a general structure that resembles a traditional SQL system catalog, as Figure 7 shows. Furthermore, we can't predefine a table or array representing the actual data, because the logical schema of the FITS files appears only when the repository attaches. Hence, an additional operation during the attachment of a FITS file repository creates the database tables representing the external file tables for querying purposes. The `cfitsio` library (http://heasarc.gsfc.nasa.gov/fitsio) extracts (meta)data from FITS files.

Query processing. After the application attaches an external FITS repository, the user can explore its content without touching the data through simple SQL queries to the data vault catalog. For example, Query 1 in Figure 8 only lists all the tables available in the repository, while Query 2 shows all of the columns storing measurements in arcseconds and their respective tables.

```
-- Query 1
SELECT * FROM fits.tables;

-- Query 2
SELECT ft.name as table_name, fc.name as column_name, fc.units
FROM fits.tables ft JOIN fits.columns fc ON ft.tableid = fc.tableid
WHERE fc.units LIKE '%arcsec%';

-- Query 3
CREATE TABLE xmatch (
  sdss_id  BIGINT,    sdss_brightness  FLOAT,
  first_id BIGINT,    first_brightness FLOAT,   distance FLOAT );
INSERT INTO xmatch
SELECT sdss.objid, sdss.sky_r / 3631000,
  f.seqno, f.fInt,
  3600 * degrees( 2 * asin ( 0.5 * sqrt ( power ( sdss.cx - f.cx, 2) +
                                          power ( sdss.cy - f.cy, 2) +
                                          power ( sdss.cz - f.cz, 2) )))
FROM sdss, fits.first as f
WHERE
  3600 * degrees( 2 * asin ( 0.5 * sqrt ( power ( sdss.cx - f.cx, 2) +
                                          power ( sdss.cy - f.cy, 2) +
                                          power ( sdss.cz - f.cz, 2) ))) < 30;
```

Figure 8. FITS queries. After the application attaches an external FITS repository, the user can explore its content by simple SQL queries to the data vault catalog without touching the data.

Query 3 illustrates integrated processing over source data of different formats. In this scenario, a researcher must cross-match two astronomical surveys—Faint Images of the Radio Sky at Twenty Centimeters (FIRST; http://sundog.stsci.edu) and Sloan Digital Sky Survey (SDSS; www.sdss.org). The surveys contain information about objects in overlapping areas of the sky. To compute properties, such as the object spectral index, the scientist must match objects based on their spatial coordinates and correlate other properties measured in the surveys for different frequencies. Assume that the SDSS catalog is stored in a table in the local relational database, while the FIRST dataset is available as a table in a FITS file. After the repository attaches, the FIRST dataset appears in the database as `fits.first` table.

Query 3 computes a simplified version of surveys' cross-matching operation. It uses the Cartesian distance between objects and stores the results in the `xmatch` table, which researchers can use in subsequent analysis.

The data vault handles the just-in-time load of the required data from the external table (`fits.first`), similar to the remote-sensing case. Additionally, if the user specifies a condition that filters rows of the `fits.first` table, the data vault can use a specialized loader that filters the rows by applying some external tool, such as the Starlink Tables Infrastructure Library Tools Set (STILTS), and provide only the relevant subset to the database.[13]

Although other software tools might cross-match, they often lack scalability. High-volume datasets might require manually splitting the task into small steps—for example, a single object or set of objects from one of the surveys processed "one spoon" at a time. Instead, the data vault application uses the most efficient DBMS join algorithms, which are more convenient and enhance performance.

All three of the use cases for science disciplines and their file formats highlight the MonetDB data vault's advantages and special features. The data vault is a work-in-progress that provides a vista on different database research challenges. Wrapping the external libraries' functionality lets us capitalize upon existing tools for in situ analysis, but this still needs careful interface design and cost modeling. Efficient symbiotic query processing requires an extensible optimizer that can detect external data and libraries. Achieving

balance between generality and usability poses another challenge. We must actively collaborate with domain scientists to develop data vault into a technology that's useful across multiple sciences.

Acknowledgments

The data vault is the collective work of the MonetDB team. We acknowledge Ying Zhang for her important work on the implementation of SciQL, Bart Scheers and Joao Sa for their contributions on the FITS data vault, and Holger Pirk for his contribution in the seismology data vault. We would also like to thank our partners in the TELEIOS and COMMIT projects for their fruitful collaboration. The work reported here was partly funded by the European Union's Seventh Framework Program Information and Communication Technologies Project (EU-FP7-ICT) TELEIOS, and the Dutch national program COMMIT.

References

1. T. Hey, S. Tansley, and E.K. Tolle, *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, 2009.
2. J. Gray et al., "Scientific Data Management in the Coming Decade," *SIGMOD Record*, vol. 34, no. 4, 2005, pp. 34–41.
3. Y. Zhang et al., "SciQL: Bridging the Gap between Science and Relational DBMS," *Proc. 5th Int'l. Database Eng. and Application Symp.*, ACM, 2011, pp. 124–133.
4. E. Stolte et al., "Scientific Data Repositories: Designing for a Moving Target," *Proc. 32nd Ann. Conf. Sigmod*, ACM, 2003, pp. 349–360.
5. K. Kunchithapadam et al., "Oracle Database Filesystem," *Proc. 30th Ann. Conf. Sigmod*, ACM, 2011, pp. 1149–1160.
6. *ISO/IEC 9075-9:2008, Information Technology—Database Languages—SQL—Part 9: Management of External Data (SQL/MED)*, Am. Nat'l Standards Inst. (ANSI), 2008.
7. P. Baumann et al., "The Multidimensional Database System RasDaMan," *SIGMOD Record*, vol. 27 no. 2, 1998, pp. 575–577.
8. The SciDB Development Team, "Overview of SciDB: Large Scale Array Storage, Processing and Analysis," *Proc. 39th Ann. Sigmod*, ACM, 2010, pp. 963–968.
9. M. Ivanova, M.L. Kersten, and S. Manegold, "Data Vaults: A Symbiosis between Database Technology and Scientific File Repositories," *Proc. 24th Int'l Conf. Scientific and Statistical Database Management*, LNCS 7338, Springer, 2012, pp. 485–494.
10. Y. Kargin et al., "Instant-On Scientific Data Warehouse—Lazy ETL for Data-Intensive Research," *Proc. 6th Int'l Workshop on Business Intelligence for the Real Time Enterprise*, 2012; http://oai.cwi.nl/oai/asset/19947/19947B.pdf.
11. M. Ivanova et al., "An Architecture for Recycling Intermediates in a Column-store," *ACM Trans. Database Syst.*, vol. 35, no. 4, Nov. 2010, p. 24.
12. *SEED Reference Manual, Standard for the Exchange of Earthquake Data*, Incorporated Research Inst. for Seismology (IRIS), 2010; www.fdsn.org/seed_manual/SEEDManual_V2.4.pdf.
13. M. Taylor, *STILTS—Starlink Tables Infrastructure Library Tool Set Version 2.5*, user note, Oct. 2011; www.star.bristol.ac.uk/~mbt/stilts/sun256.pdf.

Milena Ivanova *is a core eScience engineer at the Netherlands eScience Center. Her research interests include in-database support for array processing, transparent access to external scientific file repositories, and reuse of intermediates to improve performance of workloads with commonalities. Ivanova has a PhD in computer science (with a specialization in database technology) from Uppsala University. Contact her at m.ivanova@esciencecenter.nl.*

Martin Kersten *is a Centrum Wiskunde and Informatica research fellow and a full professor at the University of Amsterdam. His research interests include architectures for parallel database management systems, performance of database systems, multimedia database applications, and data mining. Kersten has a PhD in mathematics and computer science from Vrije Universiteit Amsterdam. Contact him at Martin.Kersten@cwi.nl.*

Stefan Manegold *is the leader of the Database Architectures Research Group at Centrum Wiskunde and Informatica. His research interests include database architectures, query-processing algorithms, and data-management technology for data-intensive scientific discovery, with a focus on optimization, performance, benchmarking, and testing. Manegold received a PhD in computer science from the University of Amsterdam. Contact him at Stefan.Manegold@cwi.nl.*

Yagiz Kargin *is a doctoral candidate in the Database Architectures Research Group at Centrum Wiskunde and Informatica. His research interests include database systems, data ingestion, and scientific data management. Kargin has an MSc in computer science from Saarland University. Contact him at yagiz.kargin@cwi.nl.*