Computing in SCIENCE & ENGINEERING

www.computer.org/cise

Vol. 10, No. 1
January/February 2008

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

IEEE computer society

© 2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

For more information, please see www.ieee.org/web/publications/rights/index.html.

The sqlLoader Data-Loading Pipeline

Using a database management system (DBMS) is essential to ensure the data integrity and reliability of large, multidimensional data sets. However, loading multiterabyte data into a DBMS is a time-consuming and error-prone task that the authors have tried to automate by developing the sqlLoader pipeline—a distributed workflow system for data loading.

Database management systems offer considerable advantages and are essential for reliable data warehousing and integrity, not to mention acceptable data retrieval performance and data mining capabilities. Of course, we must ingest the data into the DBMS before we can realize any of these benefits. Loading large multidimensional data sets such as the Sloan Digital Sky Server (SDSS) into a DBMS is a critical and time-consuming step that's fraught with potential missteps. In many ways, loading the data is the weakest link in archive publishing: it's typically the most error-prone, time-consuming yet least-planned and budgeted part of archive operations.

The virtual observatory (VO)[1] effort involves integrating many historical and current data sets into online repositories. Loading these data sets (and reloading them as schemas change) is an ongoing task. With the multiterabyte-to-petabyte data volumes anticipated for upcoming archives in the VO, data-loading time is expected to become a critical factor. Disk speeds won't keep up with the exponential increase in data volume, and with the limited operational resources typical for scientific archives, loading such large archives can take several days if parallelism is used—several weeks if not. In practice, archive data often must be reloaded several times even before it's published; errors in the data and in the data-processing pipeline cause reloads, so any changes to the schema or performance considerations sometimes make reloading the data the most expedient and clean option.

The efficiency and reliability of the data-loading process therefore is of paramount importance so as to minimize the operational resources required to reload data and maximize the chance of the data being correct the first time it's published. A crucial fact about public archives is that once the data is published, it's immutable—that is, it can't be retracted or changed, especially after scientific research has been based on the data and papers have been published. As such, we get only one chance to get the data loaded correctly, so the loading procedure must be as reliable, thorough, and automated as possible.

Accordingly, we've developed a data-loading pipeline for the SDSS data called the sqlLoader.[2] We subdivided the loading process into a sequence

1521-9615/08/$25.00 © 2008 IEEE
Copublished by the IEEE CS and the AIP

Alex Szalay and Ani R. Thakar
Johns Hopkins University
Jim Gray
Microsoft Research

of steps and developed a largely automated workflow system that executes these steps (in parallel where possible) on a distributed cluster of load servers. Loading the multiterabyte SDSS archive would be chaotic, if not impossible, without the automation and workflow management that the sqlLoader pipeline provides.

Loading SDSS Data

There are three main operational stages in SDSS data loading:

- *Converting Flexible Image Transport System (FITS) binary data to ASCII comma-separated value (CSV) files*. We store the output from the SDSS processing pipelines in binary form in the operational database (OpDB). We then export it from the OpDB into binary FITS files—a standard astronomy format for binary data and images. The FITS data must be converted to ASCII CSV format before it can be ingested into the databases (see Figure 1).
- *Loading CSV files into temporary SQL Server databases*. Once we convert the data to CSV, we load it into temporary SQL Server databases called task databases and validate it—that is, we check the export for any inconsistencies and errors.
- *Publishing the data from the task databases to the final SDSS database*. This is the stage in which we merge all the data from multiple task databases and write it to the database that will be available to users, also known as the *publish* database. This stage creates the database indices and other ancillary tables. This last part is complex and time-consuming enough that we consider it a separate "finish" substage.

The first stage is performed on the Linux side where the FITS files are hosted. The last two stages are automated on the Windows side in the sqlLoader workflow as the load and publish/finish stages.

FITS-to-CSV Converter

When we export the raw data stored in the SDSS OpDB to FITS files, several different types of files correspond to distinct data sets within the SDSS data, including image and spectroscopic data. A unit of exported image data is a *chunk*, or a single resolve operation in the OpDB. Because the data is often recalibrated as the image-processing pipeline is refined, different calibrations result in different skyVersions of the image data:

- *Target* skyVersion is the calibration from which we chose the spectroscopic targets.
- *Best* skyVersion is the latest, best calibration of the data.
- *Runs* skyVersion is a temporary calibration that might be reclassified as Target or Best later. Until then, it's only for use in internal collaboration and isn't released to the public.

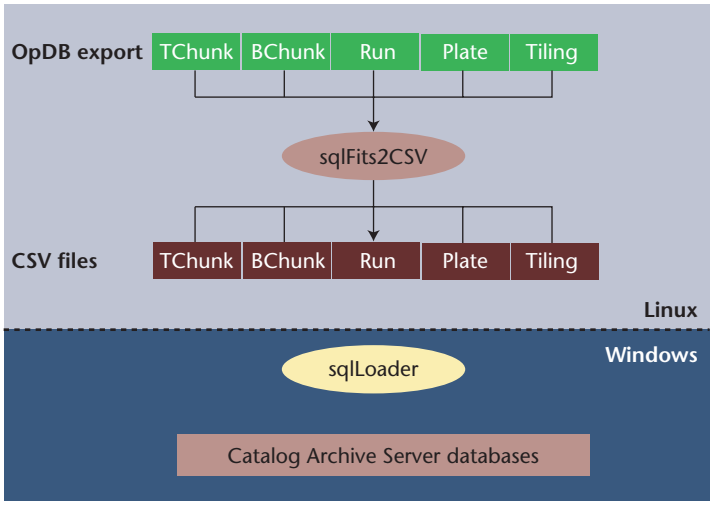


Figure 1. Sloan Digital Sky Survey data-loading pipelines. Flexible Image Transport System (FITS) data exported by the operational database (OpDB) is first converted to comma-separated value (CSV) format on the Linux side. It's then Samba-mounted and loaded into the SQL Server databases on the Windows side.

In addition to the image data sets, the public data release also includes spectroscopic plates and tiling data. A utility called sqlFits2Csv converts these FITS files into ASCII CSV files and JPEG image files for ingestion into the SQL databases. sqlFits2Csv creates one type of output file for each table in the database; it also assigns to each catalog object a unique 64-bit object ID that we use as the primary key in the corresponding database table.

The conversion to CSV also acts as a "blood–brain" barrier between the Linux and Windows worlds and between the file and database worlds—that is, it keeps either side protected from operational (as well as some functional) problems on the other side. All the popular database systems support generic CSV file converters. The CSV files are Samba-mounted on the Windows side and imported into the databases by the sqlLoader's distributed workflow system. Figure 1 shows the data flow.

Workflow Management

The SQL Server agent process controls the loader

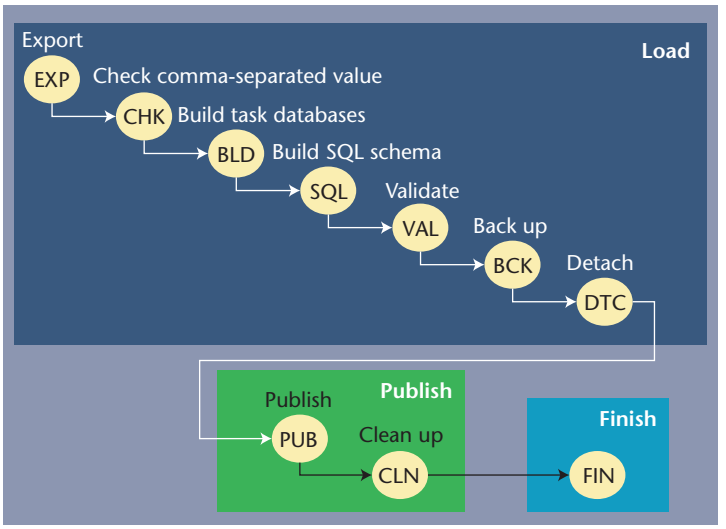


Figure 2. The workflow diagrams for the sqlLoader, showing the load, publish, and finish workflows. The load workflow is the most time-consuming and can run in distributed parallel mode. The publish and finish stages must be sequential.

workflow using two agent jobs: *load* and *pub*. The load job runs once every minute, picking up the next step from the load stage for the first active task and executing it. Similarly, the pub job wakes up once every minute and picks up the next step in the publish/merge or finish stages. Figure 2 shows the workflow stages.

The loader workflow has three distinct stages:

- The *load* stage includes checking the input CSV files, loading them into temporary task databases, validating the data in the task databases, and making a backup of the task database.
- The *publish* stage involves publishing the individual task databases to the publish database, merging the various data streams (imaging, spectro, and tiling) in the publish database, and creating the indices on the main tables.
- The *finish* stage includes running the final tests and creating the derived tables and precomputed joins as well as the corresponding indices.

We subdivide the publish stage further into two substages: *publish* and *finish*. At the moment, we must run the publish and finish stages sequentially because they write to the same (publish) database. The load/validate stages can run in parallel on a cluster of load servers, each having its own view of the schema and its own task database. On remote load servers (slaves), the local SQL Server agent also controls the workflow using account delegation. We create a domain "loadagent" account for this purpose.

Distributed Framework

There are two main parts of the sqlLoader framework—*loadadmin* and *loadsupport*—facilitate distributed and parallel data loading with a cluster of load servers. The loadadmin scripts and database control the load framework on the master load server. The loadsupport scripts and databases set up the loading on the slave load servers and build the ancillary framework to facilitate the loading process, such as setting up user privileges, the links between master and slave load servers (if applicable), spatial indexing, and utilities to manage units of the loading process (phase, step, task, and so on). Figure 3 shows the relationship between the loadadmin and loadsupport parts of the framework. Each of the satellite or slave servers links only to the master, not to each other.

The loadsupport database contains read-only remote views of the loadadmin database rather than copies so that changes in the master data or schema are automatically reflected in the slaves. This feature uses the SQL Server's *linked server* mechanism to make remote databases and tables appear to be local.

Load and Publish Roles

The loadadmin and loadsupport servers have different roles in the distributed implementation: the loadadmin server runs both in load and publish roles, whereas the loadsupport servers run only in the load role. The load role corresponds to loading CSV files into individual component databases (one per load unit), whereas the publish role involves validating and transferring the component database's contents into the publish database for each data set. Therefore, we can do the loading in parallel, but the final part of the publishing stage that brings all the loaded components together into the publish database must be done sequentially. We refer to this final publish step as the merge/finish step to avoid confusing it with the copying of the task databases to the publish database.

Several load servers can be in the load role, but only one server can be in the pub role at a time. On each slave server (master and slave can coexist on one server), the SQL agent load job picks up the next load task to execute from its view of the task table. Similarly, on the publish server (usually the master), the SQL agent pub job picks up the next publish task from its view of the task table.

Loader Command Scripts

The sqlLoader framework contains several kinds of scripts:

- Windows command scripts (.bat files) create the loading framework and invoke the SQL scripts;
- SQL scripts (.sql files) contain the SQL code to set up the loading framework and load the data;
- Data Transformation Services (DTS) packages perform special-purpose data import tasks; and
- Visual Basic (VBScript) scripts check the CSV files' syntax and parse the documentation files to load them into the databases.

The initial creation of the loader framework, including building the loadadmin and loadsupport databases, must be done via two main Windows batch/command scripts—one each to build the loadadmin (master) and loadsupport (slave) frameworks. Although for conceptual clarity we show the master hosting only a loadadmin database in Figure 3, in practice, the master server hosts both the loadadmin database and a loadsupport database.

Loader SQL Scripts

The command scripts invoke the loadadmin and loadsupport SQL scripts to perform the actual database tasks (see Table 1). We coded most of the functions required for data import, validation, and publishing in the form of SQL stored procedures in the loadsupport task and publish databases. These are defined in Table 1 and installed in the appropriate databases upon creation.

Schema Files

The schema creation scripts for the Catalog Archive Server (CAS) databases are in the schema subdirectory. The contents of this subdirectory encapsulate the data model for the archive and hence will change the most when adapted to other (non-SDSS) archives. Five subdirectories at this level contain various aspects of the database schema:

- *csv* contains CSV outputs from the documentation generation scripts—in general, this directory contains input files for metadata tables. It also contains CSV input files for tables of non-SDSS data required to do meaningful science with the SDSS (such as Stetson or RC3 catalogs) and derived science data tables such as the Quasi-Stellar Object (QSO) catalogs.
- *doc* contains the schema documentation files.

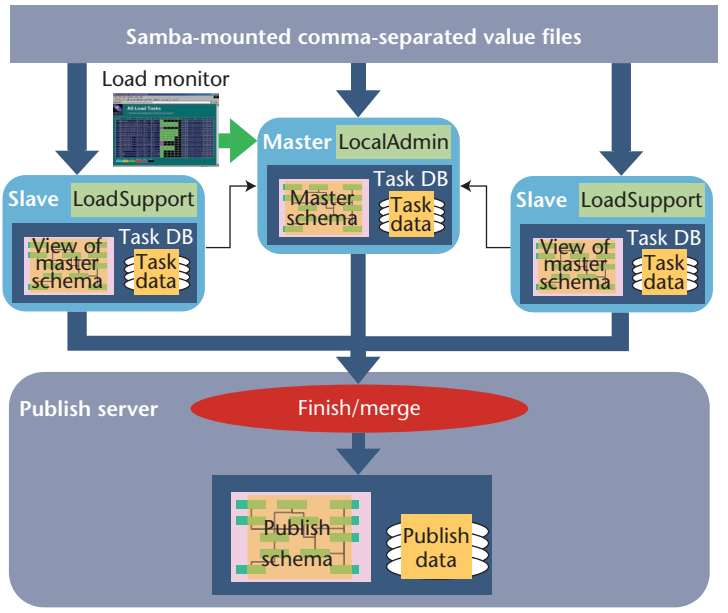


Figure 3. Master (loadadmin) and slave (loadsupport) servers. The loadsupport servers have remote views of the loadadmin schema via linked server connections. Note that the master server hosts, in practice, both loadadmin and loadsupport environments.

- *etc* includes miscellaneous SQL script files for housekeeping and utility schema-related functions.
- *log* holds the weblogging schema and scripts .
- *sql* is the main subdirectory containing the SDSS database schema files. The SQL scripts in this subdirectory create the various schema tables, views, stored procedures, and functions. The `xschema.txt` file in this directory drives the schema creation. It contains a list of schema files that the loader executes in the order indicated in Table 2.

Data Validator

Data validation is a critical step in the data-loading process. The validation procedure has the following objectives:

- *Correctness* ensures that the data-loading process and SDSS data-processing pipeline don't have errors. An important function is to uncover data corruption that can occur as a result of disk failures or any problems with networking hardware.
- *Completeness* checks that all the data is loaded and no data is lost.
- *Self-consistency* checks referential integrity.

Data emerges from the data-processing pipe-

Table 1. sqlLoader scripts required for data import, validation, and publishing.

Script file	Script functions
loadadmin-build.sql	Deletes the existing loadadmin database if any. Turns on the trace flag 1807 (a secret Windows flag that lets us mount remote databases). Sets numerous database options. Turns autoshrink off because the performance bogs down when autoshrink tasks run in the background.
loadadmin-schema.sql	Creates Task, Step, and Phase tables in the loadadmin database. Inserts null task and step to assign system errors if everything fails. Creates the NextStep table, which drives the loading sequence by specifying the procedures for the next step. Creates the ServerState table, which lets us stop the server and thus stop processing. Creates the Constants table and puts it in all the paths. Gets the server name from a global variable. (Note the SQL Server name for the server must be the same as the Windows name.)
loadadmin-local-config.sql	Must be adapted for the local configuration before running the loader. Sets up the paths for comma-separated value files. Sets up backup paths. Sets up the loadagent user and domain so that the SQL agent can be started up.
loadsupport-build.sql	Must be edited to update the domain account names explicitly. (In the future, we will likely reorganize it to deal differently with master/slave loadservers.) Creates a webagent user account used by the Load Monitor Web interface to connect to the loadserver(s) and run loader tasks to make sure that the Web agent is a sysadmin on the loadserver and that only the master loadserver does this.
loadsupport-schema.sql	Creates tables specific to loadsupport (at the moment, only the LoadServer table that contains the current server's name).
loadsupport-link.sql	Sets up the link between the master and slave server for this slave. Creates a two-way link-server relationship between the slave and master servers. Sets up all the views of the loadadmin schema. Enables remote transactions.
loadsupport-sp.sql	Sets up the stored procedures for loading from the server. Creates constructors for phase, step, and task. Performs start/end steps (only on the loadserver). Kills a task, ensuring that log records are kept and files are cleaned up. Deletes the task database when the same task is resubmitted (with new taskID).
loadsupport-utils.sql	Stores procedures for the load stage.
loadsupport-steps.sql	Controls high-level steps: each step has a stored procedure with the name "sp<name-of-step>step" associated with it, which is the meat of the step's logic, and an "sp<name-of-step>" procedure, a wrapper that calls the "sp<name-of-step>step" procedure.
loadsupport-show.sql	Displays Load Monitor screens.

line as ASCII CSV files and image files in JPEG and GIF formats. Each batch of files is imported into a staging database where it's validated and merged. Once validated, the data is moved to the production or archive databases. Figure 4 shows the validation steps.

Photometric and spectroscopic data have different routines, but the Best, Target, and Runs data sets have approximately the same validation logic.

The validator is invoked as a set of stored procedures on a particular task database; its job is to validate that database, which it does by searching the task table using the host and database names as keys. The returned record tells the validator

- the type of validation (photo or spectro);
- the type of imaging data (Best, Target, Runs) if it's a photo job;
- the destination database; and
- a job ID that's used as a key to all future log events.

spValidate then branches to spValidatePlates or spValidatePhoto (there's also an spValidateTiles for the tiling data). When these routines complete, spValidate writes a completion message in the task table and exits. At each step, the validation routines record the test's result. The Load Monitor can watch the validation progress and as-

Table 2. Sloan Digital Sky Survey database schema Data Definition Language (DDL) files.

Group	File	Description
Metadata	FileGroups.sql	Data-partitioning tables for multivolume disk
	DataConstants.sql	Data Constants table and initialization
	ConstantSupport.sql	Support functions to display constants
	MetadataTables.sql	Metadata table definitions
	IndexMap.sql	Index definitions for all tables
	Versions.sql	Database versions
Basic schema	PhotoTables.sql	Photometry (imaging) table definitions
	spPhoto.sql	Support functions for imaging data
	NeighborTables.sql	Match and Neighbors tables
	SpectroTables.sql	Spectroscopic Data tables
	spSpectro.sql	Support functions for spectroscopic data
	TilingTables.sql	Plate Tiling Information tables
	RegionTables.sql	Sector and Region tables
	FrameTables.sql	Imaging frames for JPEG display
	Views.sql	Definitions of views on all tables
Spatial functions	spHtmCSharp.sql	C# Hierarchical Triangular Mesh (HTM) library functions
	spSphericalLib.sql	Support library for HTM functions
	spNearby.sql	Proximity search functions
Region & sector	spCoordinate.sql	Coordinate conversion functions
	spRegion.sql	Functions to compute regions
	spBoundary.sql	Functions to compute survey boundaries
	spSector.sql	Functions to compute plate sectors
Diagnostics & test	spCheckDB.sql	Various diagnostics functions
	spTestQueries.sql	Test query suite
Web interface	spDocSupport.sql	Schema browser and online documentation support functions
	spSQLSupport.sql	Web SQL query handling functions
	spUrlFitsSupport.sql	Functions to compute URLs of Flexible Image Transport System file versions of the data finish, validation, and utility functions
	spNeighbor.sql	Neighbor and Match table computation functions
	spSetValues.sql	Functions to initialize various computed columns, such as HTM IDs
	spValidate.sql	Functions for data validation (validate step in loading)
	spPublish.sql	Functions to publish data to the pub database
	spFinish.sql	Functions for finish stage processing
	spCopyDBSubset.sql	Functions to make various-sized subsets of the Best database

sess the validation's success or failure by looking at this journal (in the load database) or by looking at the job summary record.

The photo (imaging) data validator performs the following tasks:

1. It checks the uniqueness of the primary key fields.
2. It creates temporary indices to make subsequent tests run faster.
3. It then tests a series of foreign keys.
4. It checks to see if the advertised populations match the real populations.
5. It looks at the parent objects to see which child objects were deblended from which parents. It then tests to see that the number of child objects matches the count for the respective parents for the first 1,000 non-null parents.

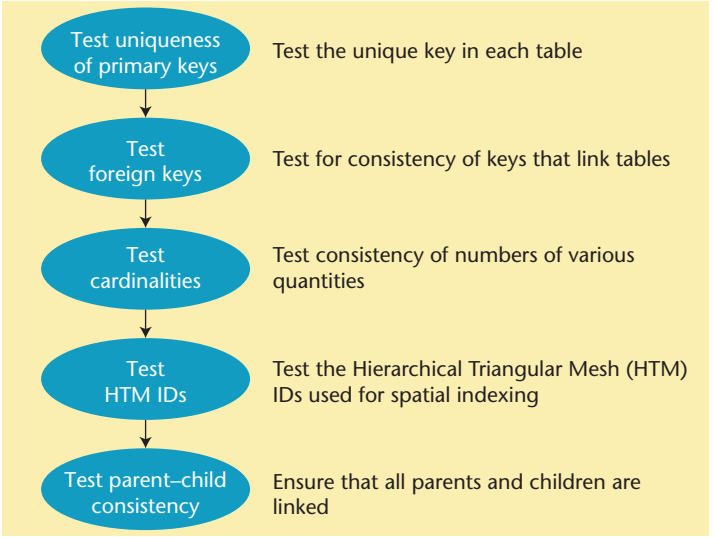


Figure 4. Data validation checks performed by the data validator. The checks range from basic data integrity tests to domain-specific self-consistency checks.

6. It also tests the first 1,000 image objects to see that the external Hierarchical Triangular Mesh (HTM) calculation is similar to the internal one. We allow a few errors due to rounding, but 99 percent of the results should agree exactly.
7. It computes the neighbors of each object; for a Best database, it computes a 30-arc-second neighborhood, but the Target and Runs databases use a 3-arc-second radius. The neighbor computation is the longest step of the validation process. The final neighbors computation in the publish database is done in the finish stage.
8. Finally, the validator drops the indices it created for the validation work.

spValidatePhoto then returns to spValidate. If the database holds spectroscopic data, it will be validated next.

Testing spectroscopic data is simpler. This data is always destined for the spectro part of the database schema, and there are fewer tests:

1. spValidateSpectro first tests the uniqueness of the primary keys.
2. It then creates two indices to make the subsequent tests run faster.
3. Next, it tests several foreign keys.
4. It also tests the first 1,000 HTM IDs in the SpecObj table (SpecObj.htmID) to check that the external HTM calculation is similar to the internal one.
5. Finally, it drops the working indices and returns.

The last stage of the loading—in which everything is brought together in the publish database, the database indices are created, and all the derived and precomputed tables are created—is complex and time-consuming enough to warrant its own section.

Finish Stage

Some of the steps in the finish process are SDSS-specific, but others are relevant for any scientific archive. The finish procedures are all enshrined in the spFinish.sql file in the schema directory, although many of these procedures call other functions and procedures elsewhere in the directory. Figure 5 shows the procedures and their dependencies.

The following steps are executed in the finish stage in order. Currently, all these steps must be executed sequentially on a single server:

1. *Drop indices.* The first step is to drop any indices as necessary before updating the data in the affected tables. For incremental loading, we usually leave the primary key (PK) indices in and drop the foreign key (FK) and other nonclustered indices.
2. *Synchronize schema.* This is usually necessary only for incremental loading, when there are schema changes since the last incremental load.
3. *Load PhotoTag.* The PhotoTag table is a vertical partition of the PhotoObjAll table and hence is created from the PhotoObjAll table during the finish processing.
4. *Build primary keys.* If any of the PKs (and associated indices) were dropped in the first step, they're recreated here. For the larger tables, such as PhotoObjAll (which is more than a terabyte for Data Release 6) and PhotoProfile (which has more than 10 billion rows), this step can take several days to finish.
5. *Build indices.* Other indices besides the PK indices are built in this step as necessary.
6. *Build foreign keys.* FKs and associated indices are built after all other indices have been built. (FK indices require the PK indices to be in place first.) The FKs on the larger tables, especially PhotoProfile, usually take several days to be created.
7. *Build Match tables.* Many objects in the SDSS

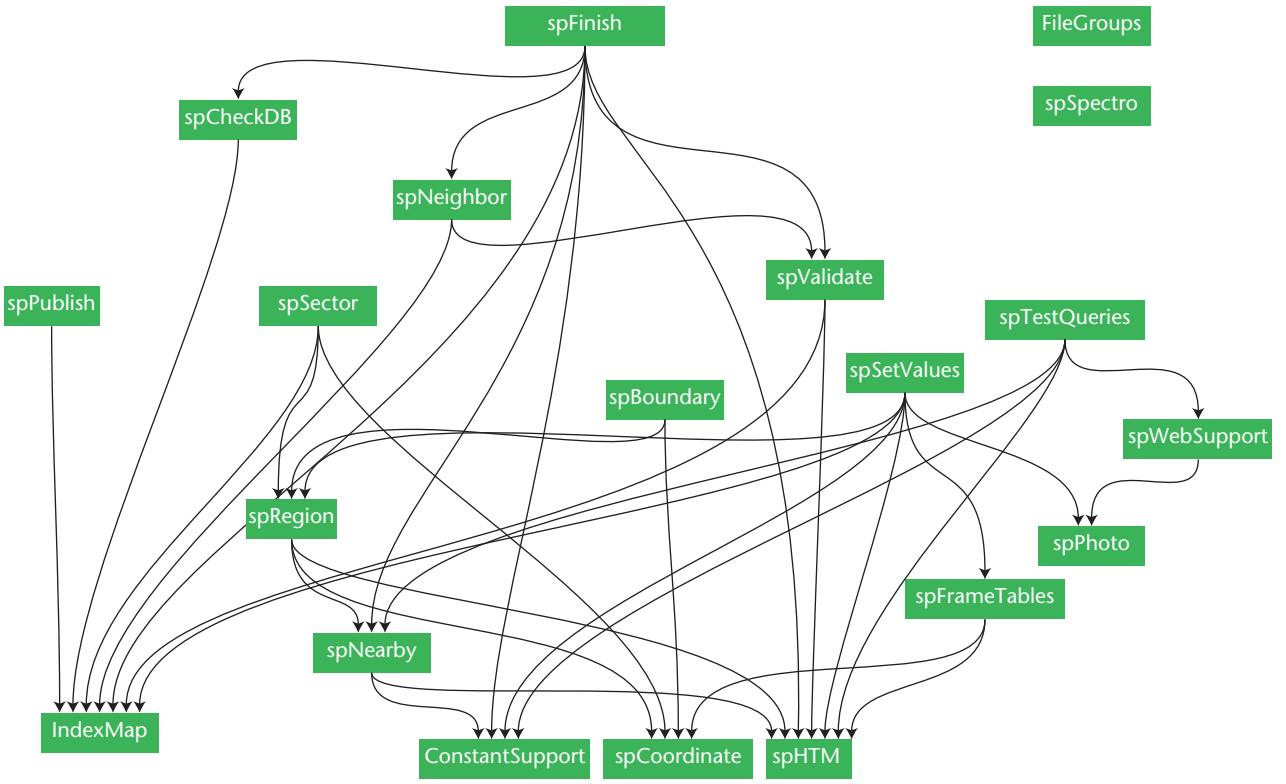


Figure 5. Dependency chart for finish functions and stored procedures. This shows the complexity and scope of the merge/finish stage in the loading; it's also the most application-dependent part of the pipeline.

data set are observed more than once, usually because they're in the overlapping boundaries between fields. The Match and Match Head tables keep track of multiply observed objects, which are useful for time-domain studies—for example, to study variable objects over time. (The procedure for building these tables is in the SkyServer Algorithms Help page at http://cas.sdss.org/dr6/en/help/docs/algorithm.asp?key=match.)

8. *Compute neighbors.* The Neighbors table is a precomputed table that finds the spatial neighbors within 30 arc-seconds of each object in the PhotoObjAll table for fast proximity-type searches in the database.
9. *Compute regions and sectors.* The Region and Sector tables are part of the Tiling group of tables that contain survey geometry constructs necessary for large-scale structure studies—for example, to gauge the completeness of spectroscopic targeting. Computing the regions and especially the sectors is a complex task, and we've written a set of stored procedures and functions to compute the tiling geometry. (We describe the algorithm in the SkyServer Algorithms Help page at http://cas.sdss.org/dr6/en/help/docs/algorithm.asp?key=sector.)
10. *Load Science tables.* These (usually external) science tables are necessary for the SDSS data, and include tables of standard stars or other well-known catalogs. Sometimes they also include derived science tables, such as those computed from SDSS data.
11. *Synchronize spectra with photometry.* The spectroscopic observations are matched up spatially with the photometry in this step.
12. *Build finish indices.* Some indices must be built at the end, after most of the finish processing completes—for example, indices on Spectro tables and Match and Neighbors tables.
13. *Match Target and Best photometry.* As we mentioned earlier, we use at least two calibrations of the data: the Target calibration from which the spectroscopic targets were chosen and the Best calibration, which is the latest, greatest calibration. This step correlates the objects from the Target skyVersion with those from the Best skyVersion.
14. *Load patches.* The last step is to apply any

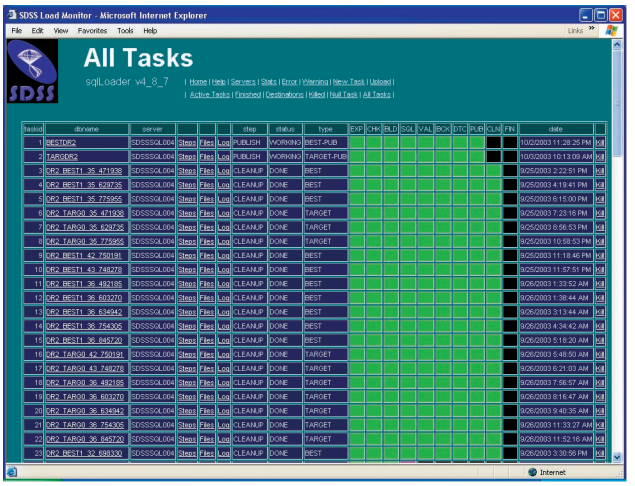


Figure 6. Load Monitor showing the All Tasks display page.

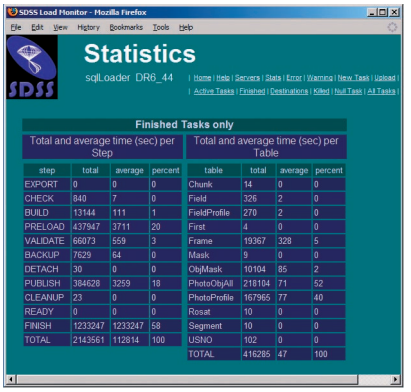


Figure 7. Load Monitor statistics page. Average and cumulative stats for all jobs are displayed.

patches necessary before releasing the data to the public. Normally, patch files are listed in a predetermined file in the schema directory and automatically loaded and executed in this step.

In addition to these steps, we must tie up a couple of loose ends manually outside the finishing process, mainly because the data isn't available during the finish. Right now, the finish stage takes more than two weeks to complete. We're looking for ways to speed this up, including parallelizing some of the steps and optimizing them for incremental data ingest.

Load Monitor or Admin Web Pages

The admin subdirectory contains the Web pages and associated files and scripts for the loader administration Web interface (the Load Monitor). This subdirectory should be copied or moved to the Web tree where the Load Monitor is to be accessed from.

The Web server connects to the loadadmin server's loadsupport database. This directory contains four kinds of files:

- active server pages (.asp) files that correspond to actual Web pages;
- Cascading Style Sheet (.css) files used by the Web pages to set up the overall look and feel;
- JavaScript (.js) files containing functions in JavaScript to perform loader admin procedures; and
- include (.inc) files, which are part of the other three file types.

ASP files correspond to each of the commands listed in the menu on the main sqlLoader page. Figure 6 gives an example of a Load Monitor ASP page for the All Tasks command. Each of these ASP scripts formulates and submits an SQL query to the loadsupport database on the loadadmin server, which sees the query as being submitted by the webagent user. The query can execute a stored procedure in the loadsupport database or request rows from a loadsupport table.

In addition to scripts for displaying the various pages showing task logs and information, there is an ASP script that displays a summary statistics page for loader tasks showing the average total time taken for each step in the task and for each table processed (see Figure 7).

Task Management

The administrator launches the loading process from the Load Monitor Web interface. The basic unit of loader processing at the top level is a *task*. A new task must be created for each unit of the loading using the New Task or Upload pages.

The loader framework divides tasks into steps and then subdivides them into phases. Steps have a well-defined start and end, but a phase does not. A step also has an SQL stored procedure associated with it. The Tasks Display pages (active, all, finished, and killed tasks) show task tables containing the task, step, and phase IDs, hence the granularity of the task display is a single phase.

Creating a new task. The Add New Task page (see Figure 8) creates a new loader task. The user must enter the following task parameters:

- The *data set* is the release being loaded (for ex-

ample, DR1, DR2, or TEST for testing). It's the same as the first parameter given to the build-publish-db.bat script.
- The *export type* is the data set to which we're exporting. The choices are Best, Runs, or Target for an imaging load; plates for spectro; tiles for tiling; and a special export type called finish for the last step that merges all the data streams (photo, spectro, and tiling) and computes the indices.
- The *xroot* is the root of the exported CSV directory tree on the Linux side (Samba-mounted). In Windows notation, this is \\hostname\directory\subdir.
- *xid* is the identifier of the export or load unit—that is, the chunk, plates, or tiles that must be loaded. This is the name of the subdirectory in the CSV directory tree that contains the runs, plates, or tiles to be loaded.
- The *user* is the name of the person running this load task.
- Optional *comments* describe this load's purpose or content.

New tasks can be added singly or in bulk. The Add New Task page is for adding one task at a time. The administrator can also upload a file containing multiple task definitions using the Upload page.

Submitting multiple load tasks (file upload). Users can submit multiple load tasks at once by building an upload file containing the task parameters values in CSV format. Here's an example of a load file's contents:

```
DR2,BEST,\\sdssdata\dr2data\csv\phCSV\
    best\1-82-1176576\,JohnDoe,best34
DR2,BEST,\\sdssdata\dr2data\csv\phCSV\
    best\1-86-1184468\,JohnDoe,best35
DR2,BEST,\\sdssdata\dr2data\csv\phCSV\
    best\1-86-1402340\,JohnDoe,best38
DR2,BEST,\\sdssdata\dr2data\csv\phCSV\
    best\1-86-1422868\,JohnDoe,best39
DR2,TARGET,\\sdssdata\dr2data\csv\
    phCSV\target\0-37-718022\,JohnDoe,
    targ39
DR2,TARGET,\\sdssdata\dr2data\csv\
    phCSV\target\0-82-1113106\,JohnDoe,
    targ40
```

Killing a task. A user can kill a task by clicking on the last column of the task display in the tasks table. After prompting the user for confirmation, the loader cleans up the information associated with that task. However, it won't delete some files, especially the temporary task databases, until the administrator submits the same task (with the same parameters but a different task ID) again. This is intentional because recreating a database is a time-consuming process, and we assume that, in most cases, a killed task will be rerun at a later date, so the data must be loaded into the database eventually. Of course, the administrator can always manually delete the task database in SQL Server after the task is killed.

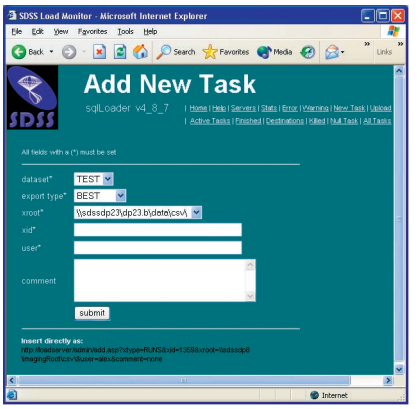


Figure 8. Load Monitor Add New Task page. New tasks can be added individually or uploaded in a text file containing multiple task specifications. This is the page for adding a single new task.

Monitoring the load. Selecting the active tasks or all tasks links in the Load Monitor shows users the tasks currently running (see Figure 6). The color coding for the task status is shown below the task table. For each task, the task ID, the step ID, and the phase number are shown, along with the name of the task and step currently executing. The task display updates once every minute.

For each task, users can select the steps, files, or log links to look at the steps, files, and phases logged (completed) for that task.

The preload step of a loading task usually takes the longest time because CSV files are loaded into the load database in this step. The largest CSV files for each run—the PhotoObjAll*.csv files—will take 10 to 15 minutes each to load, and the preload step for one imaging chunk can take more than an hour to complete. Users can monitor the step's progress by selecting the display for that task.

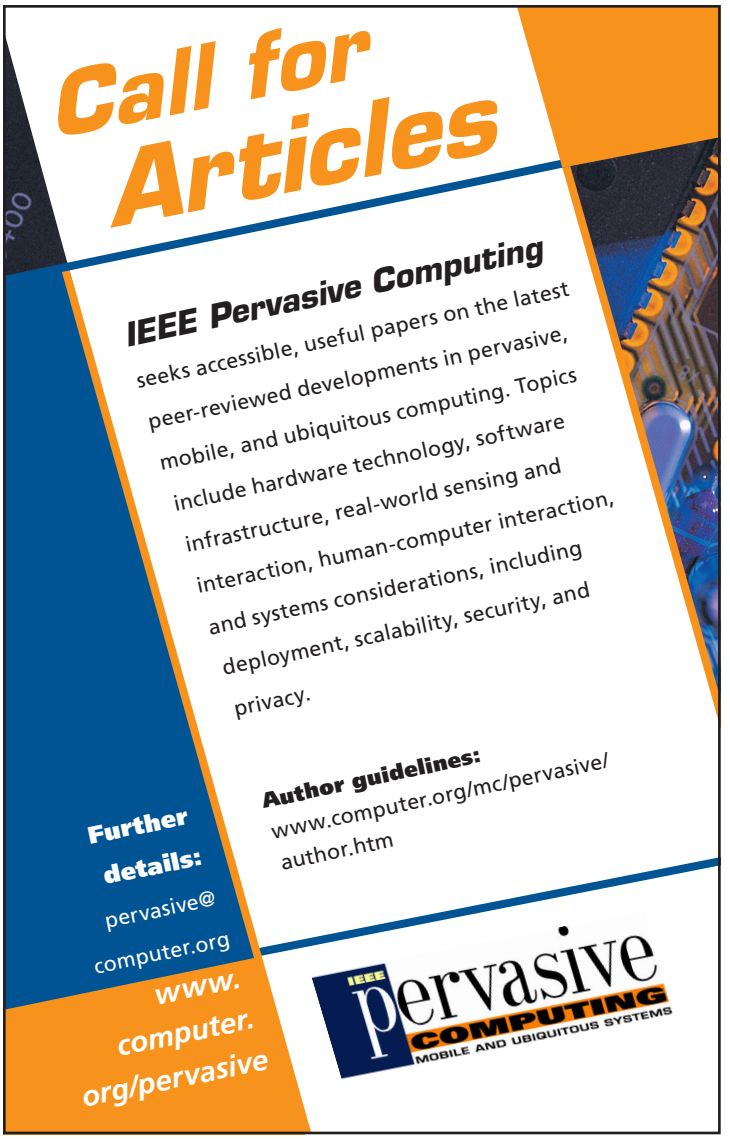
The Load Monitor also lets the administrator start and stop the SQL Server on each of the loadservers from the Servers page.

As we continue to use the sqlLoader to load SDSS-II (the successor to SDSS) data and look toward future archives, we're concentrating on several areas of future development. First, several steps in the finish stage recreate tables from scratch even when data is incrementally added to the existing archive. Most of these would run in a fraction of their current time if we could adapt them to work incrementally.

We're also looking to fully automate every aspect of the pipeline. Parts of the loading process still require human intervention, especially when things go wrong. Precious time is lost when this happens, so the pipeline needs to be fully automated.

Another crucial goal is to make the process scalable to petabyte-scale archives. This entails data partitioning across a cluster of data nodes so we can scale the loader out to archives that are an order of magnitude or more larger than SDSS, such as the Panoramic Survey Telescope and Rapid Response System (Pan-STARRS, http://pan-starrs.ifa.hawaii.edu/) and the Large Synoptic Survey Telescope (LSST, www.lsst.org/).

Lastly, we're researching ways to generalize our current framework. The sqlLoader is tightly coupled to the SDSS schema, even though we've tried to keep all the schema files in a single directory. More work is needed to decouple the schema from the framework. In doing so, we also want to extend the pipeline to other DBMS environments beyond the SQL Server. This is the most challenging task, but it's essential if the sqlLoader pipeline is to become a general-purpose data-loading pipeline for other archives in astronomy and beyond. CiSE



References

1. A. Szalay, "The National Virtual Observatory," *Proc. Conf. Astronomical Data Analysis Software and Systems* (ADASS) *X*, vol. 238, F.R. Harnden Jr., F.A. Primini, and H.E. Payne, eds., Astronomical Soc. of the Pacific, 2001., p.3.
2. A. Thakar, S. Szalay, and J. Gray, "From FITS to SQL - Loading and Publishing the SDSS Data," *Astronomical Data Analysis Software and Systems (ADASS) XIII*, vol. 314, F. Ochsenbein, M.G. Allen, and D. Egret, eds., Astronomical Soc. of the Pacific, 2004., p. 38.

Alex Szalay *is Alumni Centennial Professor of Physics and Astronomy at the Johns Hopkins University. His research interests include cosmology, large-scale structure of the universe, data mining, and science with large databases. Szalay has a PhD from Eotvos University, Hungary. He is a member of the American Astronomical Society and the Association for Computing Machinery (ACM). Contact him at szalay@jhu.edu.*

Ani R. Thakar *is a research scientist in the Center for Astrophysical Sciences at the Johns Hopkins University. His research interests include science with large databases, data mining, and simulations of interacting galaxies. Thakar has a PhD in astronomy from the Ohio State University. He is a member of the American Astronomical Society, the Astronomical Society of the Pacific, and the IEEE Computer Society. Contact him at thakar@jhu.edu.*

Jim Gray, *prior to his disappearance in February 2007, was the Turing Award-winning distinguished engineer, researcher, and manager of Microsoft Research's eScience Group in San Francisco. His primary research interests were in databases and transaction processing systems, with particular focus on using computers to make scientists more productive.*

The magazine of computational tools and methods for 21st century science

Interdisciplinary

Communicates to those at the intersection of science, engineering, computing, and mathematics

Emphasizes real-world applications and modern problem-solving

Top-flight departments in each issue!

- Book Reviews
- Computer Simulations
- Education
- News
- Scientific Programming
- Technologies
- Views and Opinions
- Visualization Corner

Peer-reviewed topics

2007		2008	
Jan/Feb	Anatomic Rendering	Jan/Feb	SSDS Science Archive
Mar/Apr	Stochastic Modeling	Mar/Apr	Combinatorics in Computing
May/Jun	Python: Batteries Included	May/Jun	Computational Provenance
Jul/Aug	Climate Modeling	Jul/Aug	High-Performance Computing in Education
Sep/Oct	Computational Wizardries	Sep/Oct	Novel Architectures
Nov/Dec	High-Performance Computing Defense Applications	Nov/Dec	Computational Astronomy

MEMBERS
$45/year
for print and online

Subscribe to CiSE online at http://cise.aip.org
and www.computer.org/cise

IEEE
IEEE computer society
AMERICAN INSTITUTE OF PHYSICS