

The AMI Database Schema

Steven Hanlon

University of Glasgow

Solveig Albrand, Jerome Fulachier

LPSC, Grenoble

Introduction

AMI, the Atlas Metadata Interface, is a generic database application, capable of interfacing to a wide variety of data models. Thus, the features provided by AMI, including a JavaserVlet-based web interface, and a web service interface with Java and C++ clients, can be used by any new AMI-backed application, speeding development.

This document provides an overview of the database schemas that are used in AMI. For more detail, the reader is encouraged to consult the [AMI Developers Guide](#)^{*}, from which most of the material here is drawn.

Principles of AMI Development

AMI development is based on a set of principles:

- The software should be independent of the particular RDBMS used.
- It should be possible to manage database schema evolution.
- The system should support geographic distribution.
- The interfaces should be as generic as possible.
- The software should not depend on a particular operating system.

These principles are connected. Each is important in ensuring that AMI is as re-usable as possible. Further, geographic distribution is considered an important goal in Atlas. The freedom to run AMI at sites supporting different operating systems and RDBMSs is helpful in achieving this goal.

AMI Self-Description Tables

In order to support a generic interface and smooth evolution of database schemas, certain conditions must be met by AMI databases.

- They must have tables that describe the entities (tables) and their relationships.
- Each table must have a foreign key which is not part of the application-specific data. (This allows for the possibility of changes to the application schema.)
- All tables have additional fields recording the creation time and time of last modification of each record.
- Application fields are limited to types that can be easily mapped to types in a variety of RDBMSs.

Identifier	Container	Contain	Type	ContainerKey	ContainKey	ContainPrimaryKey
1	dataset	dataset_extra	1	identifier	datasetFK	identifier
2	dataset	dataset_added_comment	1	identifier	datasetFK	identifier
3	dataset	logicalFile	0	identifier	datasetFK	identifier
5	dataset	dataset_keywords	1	identifier	datasetFK	identifier
6	logicalFile	db_end	3	identifier	none	identifier
4	dataset	inner_bridge_dataset_physics_group	1	identifier	datasetFK	identifier
8	physics_group	inner_bridge_dataset_physics_group	2	identifier	physics_groupFK	identifier
9	dataset_property	db_end	4	identifier	none	identifier
10	dataset	inner_bridge_dataset_properties	1	identifier	datasetFK	identifier
7	dataset_property	inner_bridge_dataset_properties	2	identifier	propertyFK	identifier
11	physics_group	db_end	4	identifier	none	identifier
13	prodsys_db_test	db_end	3	identifier	none	identifier

Table 1: The DC2 db_model table.

* http://atlasbkk1.in2p3.fr:8180/AMI_PUBLIC_WIKI/jsp/Wiki?AMI+DEVELOPER+GUIDE

The self-description tables allow the generic AMI interface to support different data models and manage schema changes. The most important table, 'db_model', describes the relationships between the application tables. By way of example, the 'db_model' table for an Atlas data challenge database is shown in Table .

The first column of this table is the primary key of the 'db_model' table itself (it is a somewhat confusing convention in the Atlas Production databases to name all primary keys 'identifier'). The 'container' column lists all database entities which are contain further entities, while 'contain' lists the contained entities. In this case, the 'dataset' entity contains six further entities, including several that, like 'logicalFile', are themselves containers. The 'type' field, implemented as a key to a table of definitions, specifies the nature of the relationship e.g. Aggregation, association etc. The final three columns indicate the fields of the containing and contained entities through which the relationship is implemented. For example, a 'dataset' is associated to a 'logicalFile' by declaring the field 'datasetFK' in the 'logicalFile' table to be a foreign key to the field 'identifier' in the 'dataset' table.

The 'db_field' table describes the fields that the tables in the AMI database have. This information is mainly used by the servlet-based web interface. An extract from a db_field table is shown in Table 2.

Identifier	Tab	Field	Type	Description	ShowType	Family	LinkClass	Rank
221	dataset	totalEvents	int(16)	Sum over partitions	0	empty		100
222	dataset	averageEventSize	float	Calculated from partitions	0	empty		100
223	dataset	totalDataSize	bigint(20)	Sum over partitions (MB)	0	empty		100
225	dataset	totalCPUTime	bigint(20)	Sum over partitions (Normalized Time)	0	empty		100
229	dataset	created	datetime	empty	0	empty		100
230	dataset	lastModified	datetime	empty	0	empty		100
340	dataset	prodsysCreated	datetime	empty	0	empty		100
341	dataset	prodsysModification	datetime	empty	0	empty		100
308	dataset	productionHistory	text	Enter here anything abnormal which happened during production	0	empty		100
337	dataset	creationComment	text	In general, notes about the contents of the data	0	empty		100
307	dataset	taskFK	int(11)	points to the task used to create the dataset	0	empty		100
306	dataset	datasetGUIDLow	int(11) unsigned	empty	0	empty		100
305	dataset	datasetGUIDHigh	int(11) unsigned	empty	0	empty		100

Table 2: The DC2 db_field table.

The columns of this table should be self-explanatory, except for the last three, which give the web interface information about actions associated with the field.

Projects and Processes

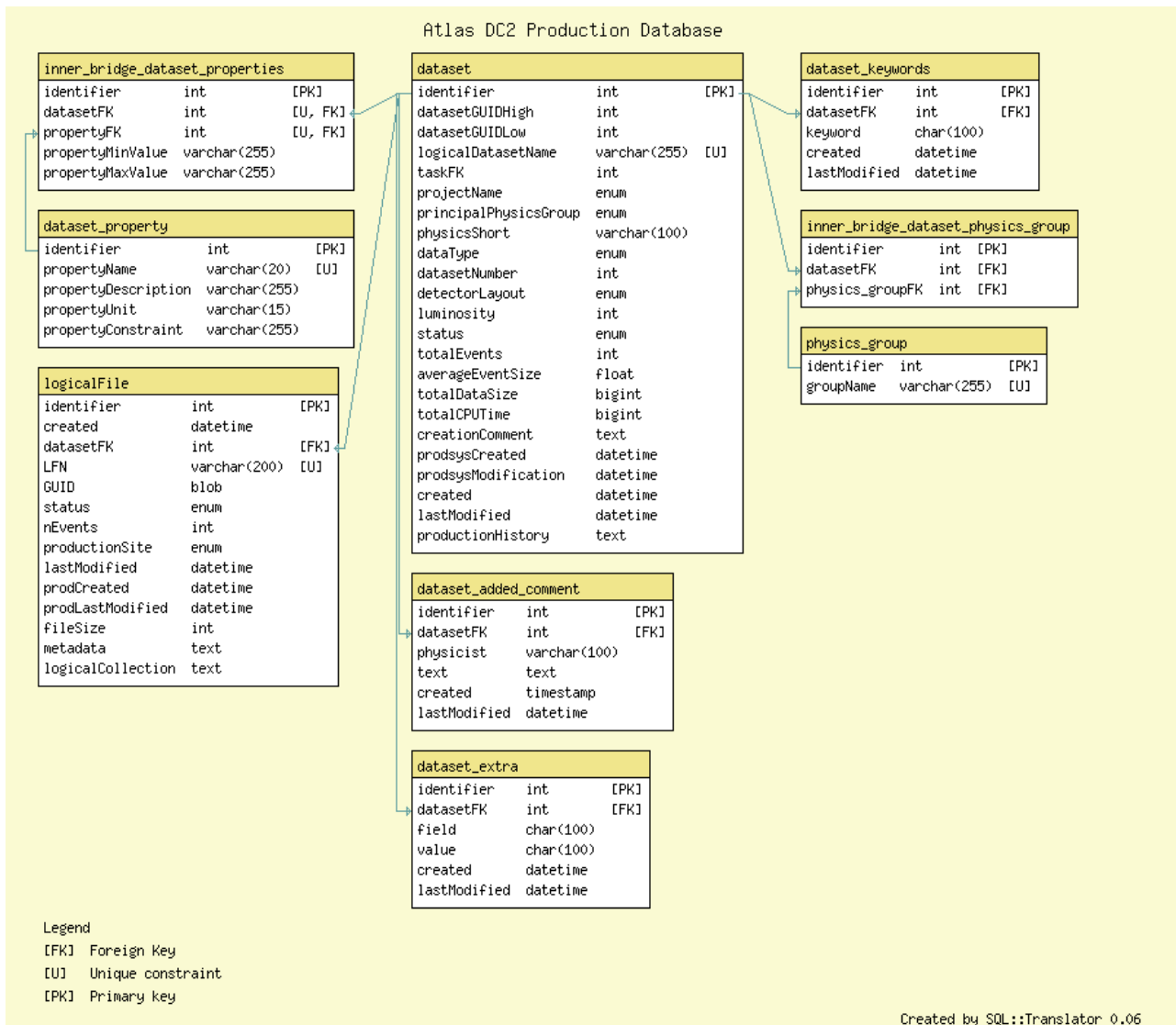
An AMI server can support multiple projects, all using different data models or even different RDBMS back-ends. It is also possible to sub-divide a project into separate blocks called 'processing steps', in reference to the Atlas data processing chain. Processing steps are separate databases (in MySQL terminology; separate schemas in Oracles terms) and can run on different RDBMS back-ends. This feature supports distribution of data, as well as allowing data that is rarely queried together to be split for performance or management reasons.

Projects and processing steps are catalogued in a router database. AMI commands sent to the server are first examined for the project and processing steps that they refer to. These are then looked up in the router database, which returns all the information necessary to contact the target database by JDBC.

The Atlas Production Databases

AMI supported the Atlas Production system for both DC1 and DC2. AMI's role has been to allow users to locate a dataset that is of interest to them using a search interface, determine which logical files it contains and discover information about those files. The file information available is that which may be of interest to a physicist. AMI does not store low-level file metadata. In particular, AMI does not serve as a Grid file catalogue, and stores no information about physical instances of logical files.

An example data model, that used for the DC2 data, is shown below.



Most of the fields in these tables should be self-explanatory, but some may require additional explanation.

'dataset'

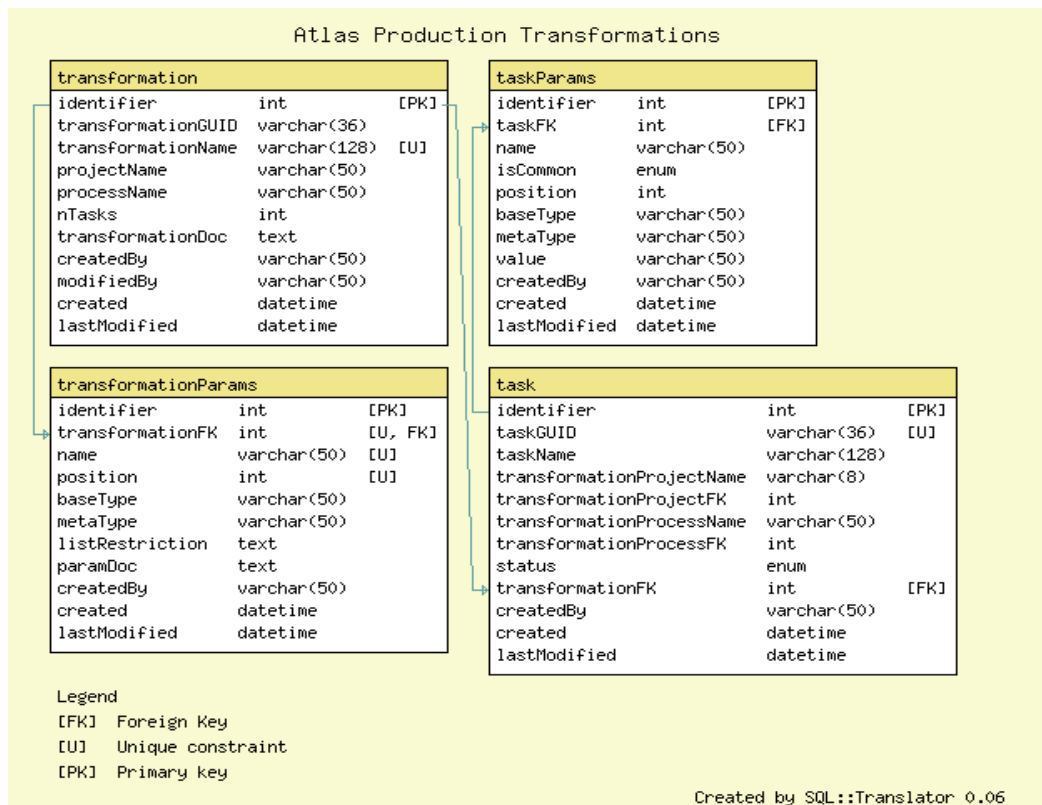
- taskFK – a key linking the dataset to the process that created it. The 'task' table is not implemented in the version of the schema shown here. This is discussed further below.
- physicsShort – a short comment. Larger comments can be added using the 'dataset_added_comment' table.
- dataType – processing stage of the data contained by the dataset i.e. AOD, ESD etc.
- detectorLayout – all the DC2 datasets contain simulated events. This field specifies the detector configuration used in the simulation.

'logicalFile'

- metadata – Atlas data challenge files are stored in **POOL*** format. This field contains XML metadata from the POOL file catalogue.
- logicalCollection – the logical path to the logical file name.

The production system also uses AMI to manage the transformations that can be applied to datasets. The schema used for this is shown below.

* <http://pool.cern.ch/>



A 'transformation' is a general form of a process that can be carried out on a dataset and has parameters that must be defined to carry it out. A 'task' is an instance of a 'transformation' with the parameters set to particular values. Both tasks and transformations can be limited to a particular project and processingStep scope.

AMI and ADA

It is anticipated that AMI will play a similar role within ADA, the [Atlas Distributed Analysis*](#) framework. ADA requires a series of catalogues for the management of datasets. A Dataset Repository stores an XML description of each existing dataset. The Dataset Selection Catalogue stores searchable metadata on the subset of datasets that have been published for wider use. The Dataset Placement Catalogue stores locations that have guaranteed that they have all the files in a particular dataset, while a Dataset Management Catalogue allows the tracking of dataset lifetimes and user claims on datasets and files. In addition, ADA includes transformations, as in the Production system.

It is clear that AMI provides the functionality needed to host these catalogues. Work has begun on adapting AMI to this application. A 'globalDatasets' table has been created. This table serves as the basis for the Dataset Selection Catalogue and includes fields that are to be common to all datasets as well as links to the projects where project-specific information can be found. In addition, a set of definitions for tasks and transformations common to the production system and ADA is being defined.

Conclusions

AMI is a response to the large number of database-backed applications needed by a particle physics experiment the size of Atlas, all with similar interface requirements. It fulfils the needs of many applications by offering a generic web service and servlet interface, through the use of self-describing databases. Schema evolution can be easily managed as the AMI application does not make any assumptions about the underlying database structure. At present the Atlas Production system relies on AMI for bookkeeping services and work to implement the Atlas Distributed Analysis model in AMI is under way.

* http://agenda.cern.ch/askArchive.php?base=agenda&categ=a051248&id=a051248s2t2%2Ftransparencies%2F050308_ada.pdf