EPJ Web of Conferences **116**, 07004 (2016)
DOI: 10.1051/epjconf/201611607004

The relational database system of KM3NeT

Arnauld Albert[1] and Cristiano Bozza[2,a] for the KM3NeT Collaboration

[1] Université de Strasbourg, Université de Haute Alsace, GRPHE, 34 rue du Grillenbreit, Colmar, 68008 France
[2] University of Salerno, Dipartimento di Fisica, via Giovanni Paolo II 132, Fisciano 84084, Italy

Abstract. The KM3NeT Collaboration is building a new generation of neutrino telescopes in the Mediterranean Sea. For these telescopes, a relational database is designed and implemented for several purposes, such as the centralised management of accounts, the storage of all documentation about components and the status of the detector and information about slow control and calibration data. It also contains information useful during the construction and the data acquisition phases. Highlights in the database schema, storage and management are discussed along with design choices that have impact on performances. In most cases, the database is not accessed directly by applications, but via a custom designed Web application server.

1. Overview

The KM3NeT Collaboration [1] is installing water-Cherenkov neutrino telescopes [2] in several sites in the Mediterranean Sea. First data are expected by the end of 2015. The detectors of the telescopes comprise many detection units, i.e. vertical lines supporting 18 regularly spaced digital optical modules; each optical module hosts 31 photomultipliers, monitoring instruments and piezo elements for acoustic positioning of the modules. The detectors are subdivided into detector *building blocks* each comprising 115 detection units with 64,170 photomultipliers in total. The complexity of tasks of the detector and the inherently multi-site nature of the challenge call for suitable data management tools. A relational database system [3] has been set up and is already providing operational functionality while it is still partially in development.

Many tables and concepts in the design of the database are inherited from the database of ANTARES [4], but substantial evolution was needed scaling performances up to meet the KM3NeT needs. This work has already started a few years ago [5]. The foundation of the system is Oracle Database Server [6]. Reflecting the geographical distribution of the telescopes, a multi-master database network is in place: a cluster of 3 servers is hosted and managed by the CC-IN2P3 centre in Lyon; another database server is hosted at the University of Napoli (UNINA), under the management of the ReCaS consortium. Two web servers, also hosted at CC-IN2P3 and at UNINA/ReCaS, are used to provide access to the databases. Both servers have interchangeable links, i.e. each one can connect to either master database.

[a] e-mail: cbozza@unisa.it

Table 1. Groups of tables in the KM3NeT database schema.

Group	Description
KM3NeT accounts	Members and external personnel, affiliation, roles, security credentials
Locations	Locations relevant to the KM3NeT research infrastructure
Institutions	Institutions and entities that have membership or other relationships with KM3NeT Collaboration
Operations	Simple or complex actions (possibly hierarchically described) that contain space and time information, e.g. construction, qualification tests, detector operation, etc.
Products	Hardware and software components of each detector and the related data acquisition and post-processing chain
Parameters	Definition of quantities that can be documented
Values	Values of parameters from datasheets, calibrations, tests, detector setup
Data streams	Streams of calibration data from quasi-on-line post-processing and from acoustic sensors (mostly used to determine the position and orientation of the optical modules and the shape of the detection units)

The database system is conceived as a living system that directly supports and takes part in several tasks, e.g. Collaboration management, detector construction and operation, data reconstruction and analysis and simulations. High performances both in writing and reading data are crucial for the whole system to be useful. System reliability and availability are very important as well for involving the database in real time data production and data collection; without these features the database would be just an offline container.

At the time of writing, the full schema of the database contains 92 tables which will not be described in detail. Instead, groups of tables that are logically linked are sketched as shown in Table 1.

2. Security

In order to ensure data security and reliable database operation, a layered security model has been adopted to provide user and batch process access. At the lowest layer, Oracle Database roles define consistent sets of privileges and permissions for access to tables, views, functions and procedures. Oracle Database accounts are provided to services with specific roles, such as the Web server and the shore station database interface. All duty accesses use HTTP(S) services, e.g. both the Web server and the shore station database interface behave as HTTP(S) servers. Physical users have KM3NeT accounts with irreversibly encrypted passwords that do not correspond to any Oracle account; and their privileges are defined in terms of KM3NeT responsibilities, e.g. membership of governing bodies, construction management or data-taking management. All KM3NeT IT services, such as the Wiki, SVN, TRAC and Google Drive access the database via the Web server as the central provider of authentication credentials.

3. Data flows

All relevant information is described in terms of *parameters*. The scalar type for each parameter is defined, and, when applicable, also the international standard unit is declared. Datasheets and tests for qualification, characterisation and acceptance are all described in terms of the set of parameters they yield. In addition, the database contains the information needed to prepare commonly used plots and graphs from the data of each dataset. This enables the Web server to automatically prepare reports in a user-friendly fashion.

The most relevant data flows in the KM3NeT database are related to detector construction, data-taking and offline event-reconstruction/simulation. The construction data flow starts with the description of purchased components; next, each object that is the result of an integration of smaller components is

described in terms of a template and the real components that were used to assemble the product; cabling information is also added when needed. All results of tests (acceptance/qualification/characterisation) on objects are documented in the database. An object that is the result of an integration operation may on its turn be part of a more complex product, e.g. several optical modules are used to build a detection unit. This implies that the database is part of a loop in the information flow. A single photomultiplier test can involve up to about 10^5 measurements, corresponding to about 10 MB; this leads to the estimate of about 1 TB/*building block* for test results.

The information collected about a complex product is later used to drive data acquisition tasks, which need the full description of the detector. In addition, for several operational parameters of the detector defaults are determined automatically from test information. For instance, the supply voltage of a photomultiplier is extracted from the characterisation data provided by the vendor; then it is first optimized on a test bench; later it is re-optimized once the detection unit is fully assembled. This last value is then used during normal data acquisition tasks to set the working point of the photomultiplier. A *runsetup* is a full set of tuneable parameters for an acquisition task. It is possible to have several *runsetups*, according to different tasks and working conditions of the detector: for example, in times of high bioluminescence one might have a different tuning of supply voltage, threshold and trigger to reduce the trigger rate. The database hosts all available *runsetups* and the timeline of all *runsetups* used during the detector lifetime, so that data analysis and event reconstruction can be tuned accordingly. The output parameters from detector monitoring and slow control are read continuously (10^{-2} Hz) and fed into the database, with an estimated data rate of a few TB/y for a detector *building block*. The detector is calibrated and its shape is recomputed continuously. The expected data rate for calibration information and for acoustic data and shape parameters also tops a few TB/year.

The KM3NeT detectors will be built incrementally: each detection unit can start taking data independently of the availability of the others, as soon as it is installed on the seabed and connected to the seabed infrastructure. Hence, the telescope as a whole has a history that requires bookkeeping.

Finally, also for the purpose of simulating the expected output of the detector, detailed knowledge of detector configurations and working *runsetups* are needed. The database is thus accessed also by simulation programs to extract such information.

4. Storage

Tables in KM3NeT database belong to either of two categories: *critical* or *non-critical*, depending on the expected cardinality in 10 years: the criticality threshold is set at about 10^6 in 10 years. Non-critical objects get a database-wide unique ID from a single character sequence. Their storage and retrieval does not require special care. The category of critical object is worth some discussion. Critical tables all contain time series of parameters or event logs. Usual heap-organized tables with several fields per record would pose several problems:

1) With 10^{12} records, adding or removing a single field is hardly feasible because the tables would be locked for more than one week.
2) Reading a single field requires reading the full row of the table, with extra disk activity and reduced access speed.
3) Accessing a time interval of values would require index scans of the primary key in addition to the table, implying sub-optimal performance.

The solution is to use a *columnar* database storage scheme; more properly, it is quasi-columnar because each record also includes the timestamp. Data tables differ only by the data type: byte, 16-bit word, 32-bit double-word, 64-bit quad-word, single-precision floating point, double-precision floating point, string or event. The primary key includes the identifier of the parameter, its source (e.g. which photomultiplier or optical module it is related to) and the timestamp; a single value field contains the

timestamped value of the parameter or the event code. Data are physically stored as if they were a stream, not repeating the unchanged key fields at every record, but storing only the timestamp and related value. This is possible by using index-organized tables with key compression. Repeated fields are skipped by physically storing the table data into the primary key. This resembles a normal data file, which contains a small header section followed by time series of data. Obvious advantages are:

1) Because identical data are not repeated, it is common to reduce the total storage size by a factor 4 or more;
2) Record access speed improves by the same factor.

5. Access via Web services

An HTTP(S) server has been written as a custom C# [7] library that can be added to any Common Language Runtime (CLR) [8] executable, so its source code is completely under KM3NeT control. It is normally run with the Mono [9] CLR on Linux, but it works with all major operating systems as well without recompilation. The Web server that provides the main access point to the database is based on the HTTP(S) server, but does much more than just providing static or dynamic Web pages for graphical user interface: it has indeed been conceived rather as an Application Web server with data management and distribution tasks. For instance, if a large (GB-size) dataset is sent for insertion into the database, a deferred insertion job is started and the result (including possible errors) is mailed later to the user that posted the dataset. Commonly accessed datasets might be automatically cached as files on disk, ready to be served to incoming requests. The cache management algorithm may consider both data size and request frequency to maximize the throughput in a proactive way. Several IT services access the Web server via *server pages* (i.e. pages not intended for human usage) to read and write data from the database. Data can be exchanged in XML or JSON format. SQL queries are replaced by HTTP GET requests with query parameters in the URL. Simple scripting tools such as *wget* and *cURL* can also be used, as well as any Web client libraries ready for inclusion in programs that need access to database information.

In the shore station, access to the database goes via the Database Interface, providing access to specific datasets through HTTP GET requests and returning data in XML format. This service also takes care of uploading detector monitoring files and calibration/acoustic data to the database. Programs that need to write to the database just have to produce intermediate files in a proper format, but they don't need to store database passwords or to depend on SQL clients to work, because the Database Interface deals with all the technicalities in a single place.

References

[1] `http://www.km3net.org`
[2] S. Adrián-Martínez et al., Eur. Phys. J. C **74**, 1 (2014)
[3] E. F. Codd, Comm. ACM. **13**(6), 377–387 (1970)
[4] A. Albert et al., Nucl. Inst. Meth. A **626**, S240–S242 (2011)
[5] A. Albert, C. Bozza, AIP Conf. Proc. **1631**, 167 (2014)
[6] `http://www.oracle.com`
[7] `http://www.ecma-international.org/publications/standards/Ecma-334.htm`
[8] `http://www.ecma-international.org/publications/standards/Ecma-335.htm`
[9] `http://www.go-mono.com`