A mySQL based ConditionsDB Implementation

Author: J.Lima
Date: November 14, 2002
Version: 0.1
Status: draft

Abstract

A ConditionsDB API implementation, based in the well known, fast and reliable mySQL RDBMS as been developed. Initialy with online systems in mind which could have problems running the Oracle server, this implementation has outperformed the Oracle implementation in all tests perfomed so far. This encoraging result has driven considerably improvements.

Contents

1 Introduction

The implementation of this alternative backend for the ConditionsDB was driven by our belief that, due to the real time demands imposed to the online software, it is important to try different implementation approaches, to test and benchmark them. Also an open source solution for the client API allows compilation in DAQ specific platforms like LynxOS.

We have chosen a MySQL based implementation because we already have a great deal of experience using MySQL and also because MySQL is a light-weight DBMS, which contrast with the other full-featured commercial ODBM'S (Objectivity and Oracle) being used for other Conditions DB implementations.

This work is being was initialy based in the Objectivity implementation's code[2]. However, since the development on that implementation has been discontinued, the efforts were concentrated in syncronizing your implementation with the new IT implementation, based on Oracle.

2 Architecture

From the architectural point of view, the design of the mySQL's ConditionsDB implemementation is structured in a layered fashion. Such layered design, with well defined interfaces, shown schematicaly on figure 1, improves code maintenance and eases the replacement of components. For instance, there is a work in progress to replace the mySQL RDBMS backend (the bottom most layer above the mySQL client library) for a Postgres based backend in order to evaluate the performance impact. For this only few classes had to be rewritten.

3 Database Schema

The API doesn't force us to use a particular database schema, so we've started the development by establishing a database schema which optimizes what we think it will be the typical online queries: queries for some particular condition or set of conditions at a given point in time. A very important requirement is that this schema should also allow a smooth scalability for a time increasing number of objects.

3.1 Database partitioning

In order to address the scalability problem the database has been split in smaller databases. For sake of clarity we shall call the main database, the database to which the client connects, the level 1 database. This database contains folder and tag information and, for each folder, the coordinates to a so called level 2 database. This second level databases contains tables with all Condition Objects information but

the actual condition data. That is, they store only the object information required for search criterias. A key that will be used to locate the actual data is kept in this database. The actual conditions data is stored in the level 3 databases. The mechanism with this two levels of indirection necessary to locate and retrieve the conditions data is shown in figure 2.

3.2 Loosely coupled tables

The concept of loosely coupled tables means that the relationships between some tables are not known to the server. The are known, however, to the client. This concept is a consequence of spliting the database as explained in previous section, and it has deep consequences on the API implementation. First of all, the queries cannot take advantage of table relationship: SQL joins and automatic referencial integrity checks are not possible. This is the price to pay for placing the tables anywere we want. Such a system, provided that carefully designed, will scale very well without need for special scalability features from the server side. This mean that we can, in principle, with this design, use almost any RDBMS backend. One can argue that the client code will be sligthy more complex. However, the code looks easier to understand and maintain than the PL/SQL procedures found in the Oracle implementation.

3.3 Table Description

This section describes the database schema. As a convention, related tables share a homonimous field.

This table is a self referenced table that implements the hierarchical folderset and folder structure. This table is the first search node and holds an id for a table coordinate that will be used to locate the data belonging to that folder.

```
mysql> describe folders_tbl;
+----------+---------------+------+-----+---------+----------------+
| Field    | Type          | Null | Key | Default | Extra          |
+----------+---------------+------+-----+---------+----------------+
| fld_id   | int(11)       |      | PRI | NULL    | auto_increment |
| fparent  | int(11)       |      |     | 0       |                |
| insert_t | timestamp(14) | YES  |     | NULL    |                |
| fpath    | varchar(255)  |      |     |         |                |
| fdesc    | varchar(255)  |      |     |         |                |
| fattr    | varchar(255)  |      |     |         |                |
| ddtype   | int(11)       | YES  |     | 0       |                |
| tbl_id   | int(11)       | YES  |     | NULL    |                |
| is_set   | tinyint(4)    | YES  |     | 0       |                |
+----------+---------------+------+-----+---------+----------------+
```

This table contains a list of all defined tags. The tags are associated both to

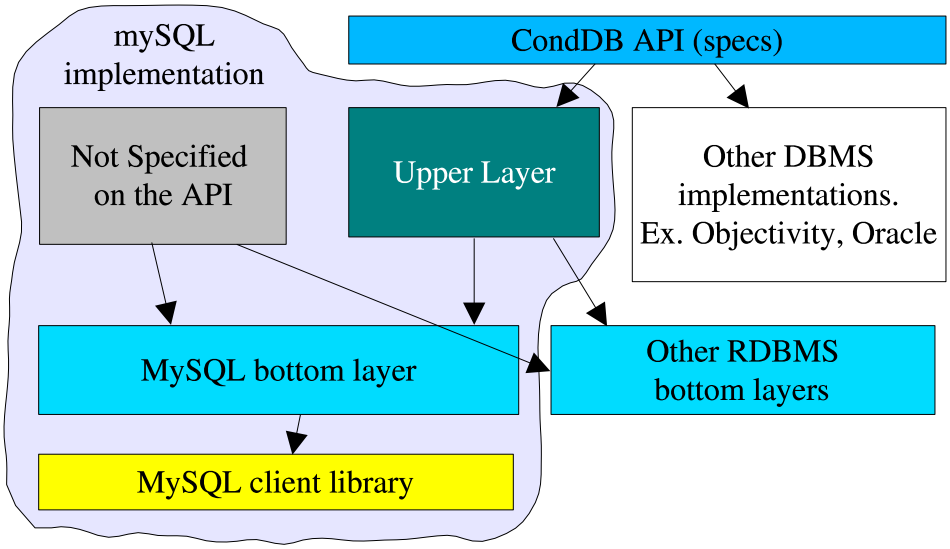

Figure 1: layered structure

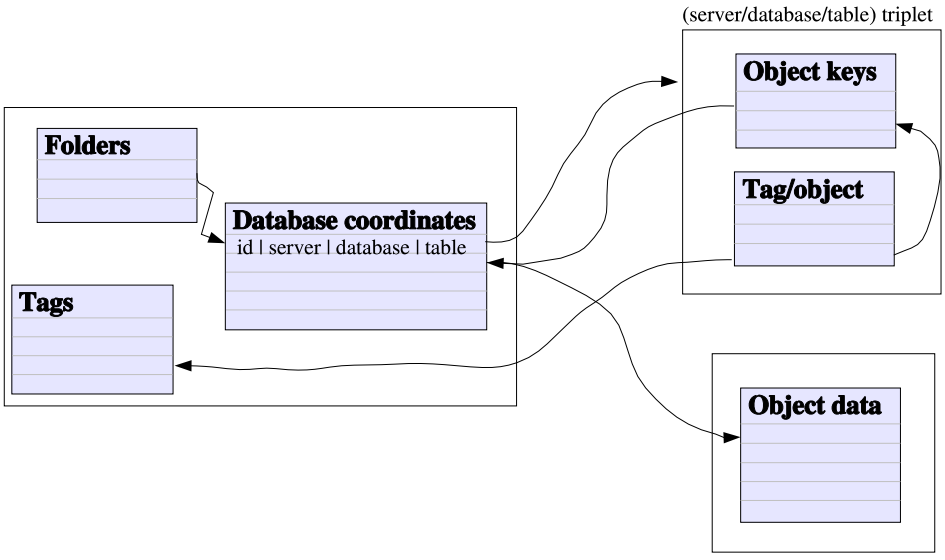

Figure 2: database schema

objects and folders. While the association of tags to objects is mandatory, the association of tags to folders is advisable for sake of efficiency.

```
mysql> describe tags_tbl;
+----------+---------------+------+-----+---------+----------------+
| Field    | Type          | Null | Key | Default | Extra          |
+----------+---------------+------+-----+---------+----------------+
| tag_id   | int(11)       |      | PRI | NULL    | auto_increment |
| insert_t | timestamp(14) | YES  |     | NULL    |                |
| tname    | varchar(64)   | YES  |     | NULL    |                |
| tattr    | varchar(64)   | YES  |     | NULL    |                |
| tdesc    | varchar(255)  | YES  |     | NULL    |                |
+----------+---------------+------+-----+---------+----------------+
```

Auxiliary table to bind tags to folders. This is a normal table relationship in the sense of the database relational model.

```
mysql> describe tag2folder_tbl;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| tag_id   | int(11)       |      | PRI | 0       |       |
| fld_id   | int(11)       |      | PRI | 0       |       |
| insert_t | timestamp(14) | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
```

Used to describe the object type, it is not yet used as the object type support is under development.

```
mysql> describe object_type_tbl;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| type_id     | int(11)      |      | PRI | NULL    | auto_increment |
| description | varchar(255) | YES  |     | NULL    |                |
| libpath     | varchar(255) | YES  |     | NULL    |                |
| code        | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

Tables to hold object table coordinates. Recall that a database coordinate is a

(server,database, table) triplet.

```
mysql> describe servers_tbl;
+---------+--------------+------+-----+---------+----------------+
| Field   | Type         | Null | Key | Default | Extra          |
+---------+--------------+------+-----+---------+----------------+
| srv_id  | int(11)      |      | PRI | NULL    | auto_increment |
| srvname | varchar(255) | YES  |     | NULL    |                |
+---------+--------------+------+-----+---------+----------------+

mysql> describe databases_tbl;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| db_id  | int(11)      |      | PRI | NULL    | auto_increment |
| dbname | varchar(255) | YES  |     | NULL    |                |
| srv_id | int(11)      |      |     | 0       |                |
+--------+--------------+------+-----+---------+----------------+

mysql> describe tables_tbl;
+---------+--------------+------+-----+---------+----------------+
| Field   | Type         | Null | Key | Default | Extra          |
+---------+--------------+------+-----+---------+----------------+
| tbl_id  | int(11)      |      | PRI | NULL    | auto_increment |
| tblname | varchar(255) | YES  |     | NULL    |                |
| db_id   | int(11)      | YES  |     | 0       |                |
+---------+--------------+------+-----+---------+----------------+
```

Default object table. up to one such table per folder can exist, although all folders can share a single table. The data partitioning must be tunned with the particular usage constraints in mind. This table doesn't contain the actual data but only the keys used for search criteria. It holds an id for the table path (tbl_id) and the id for the row within the table (dat_id) holding the actual data. Those are two

loose relationships.

```
mysql> describe def_object_key_tbl;
+----------+---------------+------+-----+---------+----------------+
| Field    | Type          | Null | Key | Default | Extra          |
+----------+---------------+------+-----+---------+----------------+
| obj_id   | int(11)       |      | PRI | NULL    | auto_increment |
| insert_t | timestamp(14) | YES  |     | NULL    |                |
| since_t  | bigint(20)    | YES  |     | NULL    |                |
| till_t   | bigint(20)    | YES  |     | NULL    |                |
| fld_id   | int(11)       | YES  |     | NULL    |                |
| layer    | int(11)       | YES  |     | NULL    |                |
| tbl_id   | int(11)       | YES  |     | NULL    |                |
| dat_id   | int(11)       | YES  |     | NULL    |                |
+----------+---------------+------+-----+---------+----------------+
```

Default data table. This table contains the actual object data, identified by a single id (dat_id) which can be easily referenced by the object table.

```
mysql> describe def_data_tbl;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| dat_id      | int(11)      |      | PRI | NULL    | auto_increment |
| data_type   | int(11)      | YES  |     | NULL    |                |
| description | varchar(255) | YES  |     | NULL    |                |
| oblock      | mediumblob   | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

Establishes the relationship between objects within a folder a a particular tag. This is a loose relationship.

```
mysql> describe def_obj2tag_tbl;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| tag_id | int(11) |      | PRI | 0       |       |
| obj_id | int(11) |      | PRI | 0       |       |
| fld_id | int(11) |      | PRI | 0       |       |
+--------+---------+------+-----+---------+-------+
```

This table defines the data partitioning policy used for a particular 2 level

database.

```
mysql> describe def_partition_tbl ;
+---------+------------+------+-----+---------+-------+
| Field   | Type       | Null | Key | Default | Extra |
+---------+------------+------+-----+---------+-------+
| tbl_id  | int(11)    |      | PRI | 0       |       |
| since_t | bigint(20) | YES  |     | NULL    |       |
| till_t  | bigint(20) | YES  |     | NULL    |       |
+---------+------------+------+-----+---------+-------+
```

4 Interface Model

4.1 Object Management

For object management we mean the operations available through the ICondDB-DataAccess interface. These include the most frequently used operations.

There are two kinds of methods to retrieve data from the conditions database: the find methods (findCondDBObject), and the browse methods, like browseObjectsAtPoint. The former returns a single object while the later return an object iterator.

4.2 Folder Management

The conditions data is organized in a filesystem like tree structure in which the nodes are folders and foldersets. FolderSets represents branches in this tree structure, while folders represent the leaves. The conditions objects (i.e the data) are stored inside folders. All the objects for all the versions and validity intervals of a given type of conditions data are grouped in a single folder. But objects belonging to different kinds of conditions data are kept in different folders.

The interface to manage folders and foldersets is provided by the ICondDB-FolderMgr class.

When creating a folder, the attribute string, formated as a XML string, will be used to determin the database where the objects will be stored.

5 Tag Management

This is the interface provided by the ICondDBTagMgr class. Provides tag creation and deletion methods. The interface allows the tagging of folders, not objects itself. When a folder is tagged, all objects at HEAD inside that folder will be tagged. Figure 3 illustrates the meaning of the HEAD and the effect of tagging.

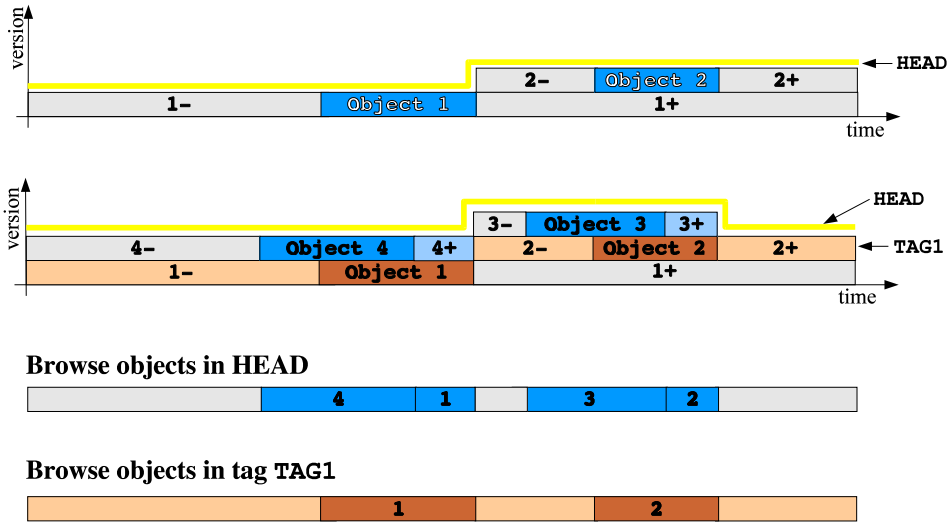

Figure 3: tagging and browsing example in the ConditionsDB mySQL's implementation.

6 Test suite and development tools

In addition to the test programs that came with the Oracle implementation and have been incorporated in our test suite, we have developed our own tests which, in principle, should give the same results on both implementations. The note[3] explains the performed tests in reasonable detail.

A ConditionsDB graphical Browser that has proven to be very useful in debuging has been developed. The browser have been used only with the mySQL's implementation but it should perform equaly well with the Oracle implementation as it only uses the API's documented methods. A browser snapshot is found in figure 4.

The browser was written in TCL using Tk and the Tix mega widget library. It interacts with the conditionsDB through a command line tool called cdbadmin which, in turn uses the C++ ConditionsDB API.

7 Development Status

7.1 Database partitioning

The database is subject to two kinds of partitioning. First, object tables are split in a per folder basis, then a time validity criteria can be used to further split the resulting object tables in smaller tables.

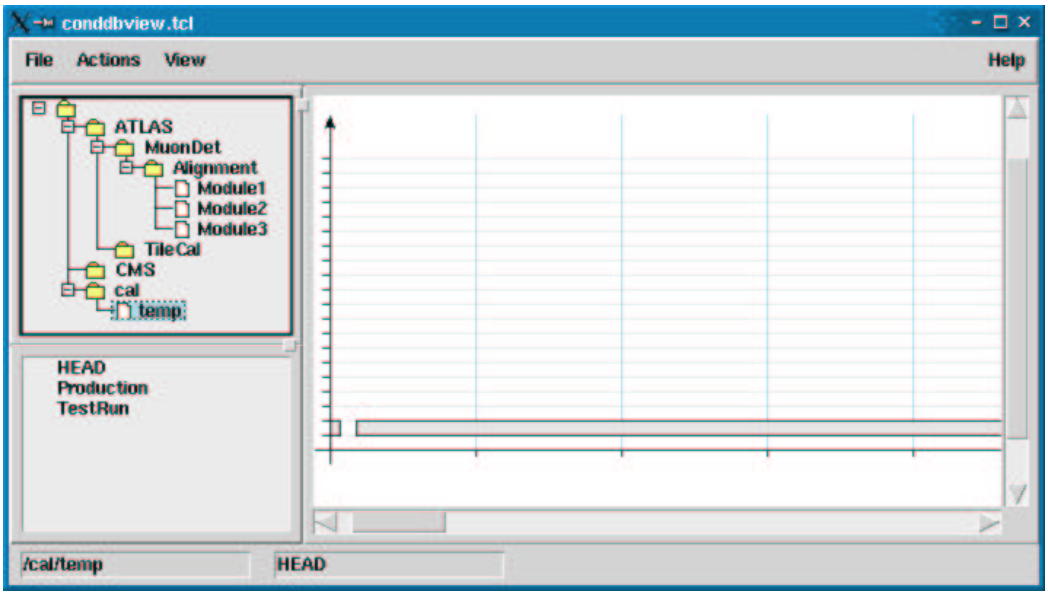

Figure 4: ConditionsDB graphical browser in action.

Status: code completely written, yet some features are not activated due to lack of a clear interface definition, namely, the creation of partitions has to be done manualy.

7.2 Specific implementation classes

This is the hierarchy of classes derived from the interface which are particular to each implementation. This implies the C++ code but not the actual queries.

Status: code completely written. All the code is pretty stable now and should need to change to make the optimizations.

7.3 Code for queries

This code hides the mySQL details and the schema details from the upper API layers. Status: completely written. However, more optimizations have to be addressed, namely exploring the usage of indexes.

7.4 Test programs and tools

Test program suite, including functional tests, performance tests, and a general purpose administration tool and browser.

Status: several tests have been written, but a more comprehensive set of tests must be designed. The general purpose administration tool and the graphical

browser are ready for expert use but should be extended and improved to achieve the desired robustness.

8 Availability

The latest version can be downloaded by ftp or http from our site at Lisbon

```
ftp://kdataserv.fis.fc.ul.pt/pub/Software/
                    ConditionsDB-MySQL-0.2.6b.tgz
http://kdataserv.fis.fc.ul.pt/ATLAS/
                    ConditionsDB-MySQL-0.2.6b.tgz
```

We have also setup a CVS repository where you can find the latest development version.

```
CVSROOT=:ext:kdata12.fis.fc.ul.pt:/usr/local/cvsroot
Package: ConditionsDB
```

Preferable, you can also access our CVS repository through a WEB interface.

```
http://kdataserv.fis.fc.ul.pt/cgi-bin/cvsweb.cgi/
```

The ConditionsDB mySQL's implementation has been placed at the ATLAS offline repository. That version is now outdated, partly because we are having trouble in setting the CMT environment. Nevertheless we expect to solve these problems and update the ATLAS offline repository soon.

```
CVSROOT=:kserver:atlas-sw.cern.ch:/atlascvs
Package: offline/Database/ConditionsDBMySQL
```

9 User Manual

9.1 Downloading the latest version

The latest version that (version 0.2.6b on time of writting) can be downloaded from one of the locations shown in previous section. If We should stress that you should never mix the databases created with some version of the code with the code from a newer version, because there are dramatic changes between versions that make databases incompatible. One of the factors that will determine a productions release is the capability to stuck with a stabilized database schema that make databases compatible.

10 Compile

Although we have only tested in two platforms, the mySQL's ConditionsDB implementation should compile on any UNIX platform. The only requierments are an working instalation of GCC (version 2.95.3 or above) with C++ support and the development files for mySQL client library (version 3.23.41 or above). Start by unpacking the distribution file in an appropriate directory.

```
tar zxf Conditions-MySQL-0.2.6b.tgz
```

Change to the source directory

```
cd ...
```

Edit the toplevel Makefile to match the settings of your host. The Makefile is heavily comented, so just follow the instructions. In the Makefile you can set the default `INIT_STRING` for every CondDB application that doesn't specify one.

Finaly type

```
make depend
make
```

10.1 Installation

The instalation settings are also defined in the toplevel *Makefile*. The default library directory is `/usr/local/lib`. You can change it to fit your needs and then, as root, type

```
make install
```

This will install the library in the appropriate place an run the ldconfig. However you'll have to set the `LD_LIBRARY_PATH` variable manualy if the defined lib directory is none of the standard system recognized ones (see next section for details).

10.2 Setting up the environment

There is a script in the utils directory which sets typical values for the environment variables. You can use it as a template and edit it to fit your needs.

The relevant environment variables are

`LD_LIBRARY_CONFIG`

This sets the library search path. This works like the PATH environment variable and should include the directory where libconddb.so is found.

For example

```
export LD_LIBRARY_PATH=/home/jmal/conddb/lib
```

COND_DB_INIT

This is the default initialization string. It is passed to the mySQL server at startup and is of the form.

```
hostname:database_name:user:password
```

The real initialization string is defined as follows: if the application provides an initialization string when calling the method CondDBmgr-¿init() the application will use that string; if, on the other hand, CondDBmgr-¿init() method is called with an empty argument list, the application will look for the initialization string in the environment variable COND_DB_INIT; Finaly, if COND_DB_INIT is not defined, the compile time default initialization string (defined in Makefile) is used instead.

COND_DB_DEBUG

This environment variable defines the level of debuging messages available when running applications, provided that the API has been compiled with debug support. From version 0.2.6a the debug messages are sent to a terminal other than the one where we run the programs (/dev/pts/0). This works on Suse-Linux but might not work on other systems. In won't work without X11, or if the /dev/pts filesystem is not activated in the kernel. You can edit the Makefile and change this behaviour. (see coments on the Makefile).

10.3 Runing the tests

The main make target builds the ConditionsDB library as well as the test programs. To perform the tests change to the tests directory an execute the programs. Some of the test programs depend on others and other programs will fail if executed twice. Although not strictly necessary to follow, we recomend the following sequence.

./basicSession Creates the database.

./storeData Stores a simple object.

./readData Readsback the object.

./genericObjectStore Stores a vector object.

./genericObjectRead Reads the vector object.

./storeDatax Stores multiple object (up to 1 million).

./readDatax Reads multiple objects.

./createTags Creates some tags.

./testTags Tag objects.

10.4 Build your own examples

The following lines can be used as a template for a bare ConditionsDB application, but we advise you to take a look at one of the simple test programs, like storeData, in the test directory.

The red lines are the only ones that will change according to the used implementation, Those are only 3 lines, no matter how long the code extends. For the mySQL's implementation 'XXXX' must be replaced by 'MySQL'.

```
#Include <ConditionsDB/CondDBXXXXMgrFactory.h>

...

ICondDBMgr* CondDBmgr =
CondDBXXXXMgrFactory::createCondDBMgr();

...
CondDBmgr->init();
    ...
CondDBmgr->startUpdate();
CondDBmgr->createCondDB();
CondDBmgr->commit();
...

CondDBXXXXMgrFactory::destroyCondDBMgr( CondDBmgr );
```

A more complete description of the test programs as well as the test result analysis can be found in the Conditions DB-test-note [3].

11 Future developments

There are several implementation issues subject to improvement here we'll name only the most important.

The database coordinate mechanism (see section ...) can be simplified to allow a more effective use of the concept. We have noticed that the triplet (table/database/server) can be replaced by a (database/server) doublet providing the same functionality at much less complexity.

Although the API mechanism that will allow us to define the data partitioning among the databases has been identified, the implementation is not yet complete.

Partly this is due to some uncertains about the directions of possible evolution of the interface specification and partly because we were much more involved with other implementation issues. This is one of the features that will be ready for the next release.

Basicaly that mechanism will allow to create a new database coordinate for each folder based in the information contained in the XML string passed as the attribute parameter, during the folder creation. The database coordinate will be used to create the database, if it doesn't exist yet, and to bind that folder to the newly created or existing database.

Other implementation issues that will require some sort of interface modification are being investigated.

An important effort has to be made to port the implementation to every possible hardware platform used at ATLAS. Currently the implementation builds and runs in the following platforms

SuSE 7.3 FreeBSD 4.5

There should be no problem in compiling with other versions of SuSE (at least from 7.0) or recent versions of RedHat, or even on SunOS. But, of course, test must be made.

At last but not least, besides the tests and comparisons that have allready been performed, we plan to develop a more comprehensive test suite that will cover, in a consistent manner, every realistic cases of ConditionsDB utilization.

References

[1] Stefano Paoli, *Conditions DB Interface Specification.*

[2] R.D. Schaffer, *ATLAS Database Basics.*

[3] C. Oliveira, *First Evaluation of the Ralational Implementation of the ConditionsDB.*

[4] A.Amorim et al, *Requirements on the Conditions Database Interface for TDAQ.*