External Products Integration Guide: PostgreSQL

rasdaman version 9.4

rasdaman Version 9.4 External Products Integration Guide: PostgreSQL

Preface

Overview

This guide provides information on how to use the rasdaman database system (in short: rasdaman) with the PostgreSQL database management system (DBMS) as base system to manage raster and relational data in an integrated way. The booklet addresses exclusively issues that are specific to the PostgreSQL configuration. Please refer to the PostgreSQL Manual, to the *Installation and Administration Guide* for information about installing rasdaman, the other rasdaman External Guides for information specific to other base systems, and to the other rasdaman guides for features of the rasdaman system, which are common to all platforms.

In this guide, you obtain information about the prerequisites for using rasdaman in conjunction with the PostgreSQL DBMS, how to create the rasdaman database in PostgreSQL and the particular configuration you require for running rasdaman based on PostgreSQL. It is then illustrated, with an example program, how to reference rasdaman specific instances

from conventional (non-raster) data in PostgreSQL, and thus enhance your conventional database applications with advanced rasdaman functionality for multidimensional arrays.

You should follow the instructions in this guide as you install your rasdaman server for interoperation with PostgreSQL.

Audience

The information in this manual is intended primarily for database administrators and, to some lesser extent, for application developers.

Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as the graphical-interactive query tool *rView*, and release notes.

In particular, current restrictions, known bugs, and workarounds are listed in the Release Notes. All documents, therefore, always have to be considered in conjunction with the Release Notes.

The rasdaman Documentation Set consists of the following documents:

- Installation and Administration Guide
- Query Language Guide
- C++ Developer's Guide
- Java Developer's Guide
- raswct Developer's Guide
- rView Guide

Table of Contents

1 Introduction

1.1 General Remarks

Rasdaman interoperates with the PostgreSQL database systems. There is a clear, natural distribution of work between the two different systems according to the data types: multidimensional data are managed by rasdaman, whereas for the alphanumeric data applications remain interfacing directly with the relational system[1]. At the bottom line, however, all data - multidimensional or alphanumeric - end up in the same physical database, thereby considerably easing database maintenance with respect to consistency, backup, etc. To this end, rasdaman makes use of the storage management facilities of the database system it is coupled to.

[1] In some disciplines, multidimensional data are referred to as *raw data* or *processed data*, depending on their status, whereas the accompanying alphanumeric data are called *meta data*.

For the purpose of this documentation, we will call the conventional database system to which rasdaman is interfaced the *base DBMS*, understanding that this base DBMS is in charge of all alphanumeric data maintained as relational tables or object-oriented semantic nets.

The interoperability strategy of rasdaman follows, as much as possible, the concepts of the individual base system on hand. Base DBMS features such as access rights and overall database organisation apply to multidimensional and alphanumeric data uniformly. Consequently, rasdaman introduces no new concepts there, but relies on the mechanisms provided by the base DBMS.

The advantage for the application developer and database administrator is obvious: it allows the user familiar with the base DBMS concepts to use rasdaman in the same way, thereby minimizing learning effort.

1.2 Version and Compatibility Statement

rasdaman has been tested successfully with PostgreSQL version 9.x under Linux kernel 2.6.22 and newer.

PostgreSQL 8.4 has been used successfully in the past, however note that “end of life” for PostgreSQL 8.4 will be reached in July 2014[2].

Although we are not aware of any problems using older PostgreSQL versions (except for a PostgreSQL 8.3 bug), use nevertheless may lead to unforeseeable effects, including possible loss of data.

Therefore, administrators are strongly encouraged to build rasdaman upon PostgreSQL 9.x.

[2] Taken from http://www.postgresql.org/support/versioning/

2 Technical Details

In this section, necessary background knowledge for the set-up of PostgreSQL for use with rasdaman is provided. The initialization procedure itself is described in detail in Section 3.

2.1 Account Management and rasdaman login to PostgreSQL

It is strongly suggested to create a new operating system account, `rasdaman`, used for administration of the rasdaman database and start-up of the rasdaman server. PostgreSQL by default will rely on the operating system login under which the rasdaman server runs, i.e., `rasdaman`. This means that each time a rasdaman server logs in to PostgreSQL, it is assumed to do so using the login name `rasdaman`.

Throughout the document on hand, this recommended configuration will be assumed.

2.2 Overall Database Structure

PostgreSQL employs the concept of a database cluster, which is managed by one server instance and can hold a number of databases, distinguished by their names. Rasdaman always connects to a particular database, not knowing about database clusters; hence, the administrator is free to arrange databases in a database cluster as s/he sees fit. In particular, the database creation script, `create_db.sh` (see below), assumes that a database cluster already exists.

Rasdaman relies on several tables which are created, initialized, and maintained by several shell the scripts and the `rasdl` utility (see Section 3).

Warning

It is essential that the rasdaman database tables remain under the exclusive control of rasdaman. The rasdaman tables should not be changed in any way by external applications. In addition, the names should not be used for other tables of the user application. Likewise, it is not recommended to use the prefix `RAS_` for tables external to rasdaman.

Undefined effects can occur if another program or person performs any kind of change to table structures or contents, including severe data loss.

2.3 rasdaman Database Name

The database name is passed to the rasdaman server via the `rasmgr.conf` configuration file, as described in the *Installation Guide*. For a local database this name might be chosen as `RASBASE`; see, however, 30.2. “Connecting to the Database Server” of the PostgreSQL manual for more alternatives, including accessing remote databases.

2.4 Database Internal Organisation

As PostgreSQL does not allow to control physical distribution of table sets in different files (like, e.g., Oracle and DB2 support), there is nothing to take into account.

Notes

Make sure that disk space is of sufficient size when inserting or updating data, otherwise undefined effects may occur depending on the PostgreSQL behavior.

For the database file size, operating system limits may apply

3 Database Creation Procedure

Following successful installation of the rasdaman software (as described in the rasdaman *Installation and Administration Guide*), perform the following steps for initialization of the PostgreSQL database structures required by rasdaman. In this Section, the steps for preparing PostgreSQL for interoperation with rasdaman are explained in detail.

3.1 rasdaman Installation

First, perform all installation steps as described in the *Installation and Administration Guide*. Upon successful completion, continue below.

3.2 Environment Set-Up

The following steps have to be performed to obtain a PostgreSQL database which can be used by rasdaman.

A couple of environment variables have to be set. For the purpose of this Guide, they are grouped into two sections: PostgreSQL specifics and rasdaman specifics.

PostgreSQL Variables

The following PostgreSQL variables have to be set for the rasdaman user.

```
export PGSQLDIR=/usr/local/pgsql
export PATH=$PGSQLDIR/bin:$PATH
export LD_LIBRARY_PATH=$PGSQLDIR/lib:$LD_LIBRARY_PATH
```

Note: Depending on your installation, variable `PGSQLDIR` may have to be adapted accordingly to point to the local PostgreSQL installation directory used.

PostgreSQL configuration

To allow for flawless communication between rasdaman and PostgreSQL, the following option needs to be set in the PostgreSQL configuration file `postgresql.conf`:

```
tcpip_socket = true
```

Note: It has been observed that this is not necessary with newer PostgreSQL versions.

3.3 Create and Delete Databases

Under PostgreSQL, creating and deleting a database / database cluster is performed with the `initdb` and `dropdb` commands; see the pertaining PostgreSQL manuals for details.

Following PostgreSQL recommendation, the database cluster supposed to hold the rasdaman database should be owned by the `rasdaman` operating system user, meaning that the cluster is generated under the `rasdaman` login and the PostgreSQL server is running under this login.

Script Support

In `~rasdaman/admin`, a script named `create_db.sh` is provided which accomplishes database initialisation.

This script assumes that a database cluster has been generated and that the pertaining server process is accessible from user `rasdaman`. The script performs creation of a PostgreSQL database named `RASBASE` does and rasdaman initialization via `rasdl`.

Running `rasdl` does not require the rasdaman server to be up.

Delete a Database

To delete a database *db*, use the pertaining PostgreSQL command as operating system user `rasdaman`:

```
dropdb db
```

3.4 rasdaman Object Creation and Manipulation

Having created a rasdaman schema, population of the database through *rasql* and the APIs will follow. See the resp. manuals of the rasdaman Documentation Set for more information on this.

4 PostgreSQL Known Issues

4.1 Transaction Warnings

Upon committing or aborting a PostgreSQL transaction (e.g., in the course of committing or aborting a rasdaman transaction, or upon closing a database or a connection) a warning is issued in the rasdaman server's log file:

```
commitTA...Warning/error in TransactionIf::abort() ROLLBACK:
SQLSTATE: 25P01 SQLCODE: -604
```

Simultaneously, the PostgreSQL server may issue one of the warnings below:

```
WARNING:  there is already a transaction in progress
```

```
WARNING:  there is no transaction in progress
```

Both warnings can be safely ignored.

4.2 Long Transaction Open

In very rare circumstances it has been observed that a PostgreSQL “begin transaction” command sometimes can take up to several seconds. Currently no reason is known for this phenomenon.

It is recommended to reuse transactions as much as possible to avoid the overhead of opening new transactions.

4.3 Disk Space

The PostgreSQL database grows as data are inserted. Make sure that sufficient disk space is available.

It seems that, when needed, parts of the database can be relocated to other file systems using symbolic links with the same names (with the DBMS server duly being shut down during this reorganization, of course); however, this has not been verified systematically yet.

5 Linking MDD with Other Data

In order to embed MDD objects and MDD collections in PostgreSQL databases, object identifiers and collection names may be used. These constitute references to rasdaman objects, which are stored in PostgreSQL tables.

5.1 Collection Names

MDD collections in rasdaman must be named. This name may then be used by an application as a reference to the MDD collection. The most typical usage of these collection names is their storage in a tuple attribute in order to reference an MDD collection which is related to the application entity described by the tuple.

This is illustrated in the following example:

Example

```
table patient
(  socialsecurityno int,
```

```
pname   varchar(20),
birth   date,
address varchar(200),
...
xrayoid varchar(200) );
```

The OID in the attribute `XRAYOID` is read by an application, in order to access the MDD collection through either the query language rasql, or the API raslib in a C++ application (which may also used embedded SQL from PostgreSQL, and issue rasql queries from rasdaman).

5.2 Object Identifiers

Each MDD object is uniquely identified in rasdaman by an object identifier. Object identifiers (*OIDs*) are implemented by the `r_OId` class of RasLib. Due to their globally (worldwide) uniqueness, object identifiers can be used for references across different databases and even across different systems.

A globally unique object identifier has three components describing

- the database system type (which is "`postgresql`" in the case of PostgreSQL)
- the name of the rasdaman database to which it belongs (which by default is "`RASBASE`"),
- the object itself within the database.

Note

The OID structure will change in future to allow for unique identification also in multi-server environments.

OIDs in the API

The OID of a rasdaman object is returned by (C++ notation):

```
r_OId& r_Object::get_oid( )
```

The OID may be used as a reference in a tuple of an PostgreSQL table by storing the OID value in a tuple attribute.

Example

```
table satelliteimages
(  acqlocation    int,
   acqdate        date,
   ...
   imagerasoid   varchar(200) );
```

The value of attribute `IMAGERASOID` has to be translated into a rasdaman OID. This translation is done by the C++ `r_OId` constructor of RasLib:

```
r_OId::r_OId( const char* )
```

The string representation for a specific OID is returned by:

```
const char* r_OId::get_string_representation( )
```

5.3 Transaction Handling

As a consequence of the architectural approach of rasdaman as an additional component on top of the base DBMS (as compared to integrating rasdaman into the base DBMS engine), operations on MDD and conventional alphanumeric data cannot be intermixed in the same transaction. Therefore, to work simultaneously with rasdaman and PostgreSQL data, an application must run a rasdaman transaction for multidimensional access and a separate PostgreSQL transaction for tabular data handling.

Both transaction types can be interleaved arbitrarily. It is not necessary to end an PostgreSQL transaction before a rasdaman transaction starts, and vice versa.

5.4 Application Example

In the following C++ example, the general structure of a raslib application with PostgreSQL code is shown which uses embedded rasdaman objects.

```
r_Database rasDB;
r_Transaction rasTA;
...                          // PostgreSQL declarations

rasDB.set_servername( rasServerName );
rasDB.open( "RASBASE" );
rasTA.begin( );

...                          // PostgreSQL initializations,
...                          // if needed

// work with rasdaman and PostgreSQL data interchangeably;
// queries can be issued both on MDD collections
// and on PostgreSQL tables
...

rasTA.commit( );
rasDB.close( );

...                          // PostgreSQL terminating code,
...                          // if needed
```