A schema for implementing the RIPE Database on a SQL engine.

João Luis Silva Damas

Database Group
RIPE NCC

Document: NewDB–1
Date: June 24, 1999

ABSTRACT

This document is a description of the proposed SQL schema for the new RIPE Database.

Contents

1 Current storage system

This section provides an overview of the current database storage system in order to provide a good perspective of the differences with the proposed structure in the new implementation.

The current Database stores objects in flat text files.

The Database has two modes of operation:

- SPLIT mode. There is a text file for every object type.
- NON SPLIT mode. All objects are stored in the same text file.

To be able to do fast lookups, these files are indexed.The indexes are hash data structures where the keys to the hash are the words appearing in the text of the object.

For each key the value is a comma separated list of offsets to the plain text files. These offsets point to the start of the object in the text files, identifying which objects contain the keys. This is slightly complicated in practice due to limitations in the size of hash values in previous versions of the Berkeley DB.

Currently these hashes are stored on disk using the Berkeley DB package.

Whenever modifications are done to the text files the corresponding index files are updated.

Object creation consists of appending the new object to the end of the relevant text file.

Object deletions are done by substituting the first two characters of the “deleted” object in the text file and removing the offset from the value of the dbm hash.

Object updates are equivalent to a deletion followed by a creation.

This means that the database must be cleaned up regularly to purge the objects marked for deletion from the text files or they will grow quite fast at the current update rate of 12000 updates per day (about 15% of the Database per month).

This process is very time consuming since it requires to process the text files completely to get rid of the objects marked for deletion, copying the “real” objects to a new file and at the same time do a full rebuild of the index hashes.

In the current production setup (SUN Enterprise 450, 2x 250Mhz UltraSPARC II CPUs + Hardware RAID 5) this process takes about 14 hours for the full set of data (split over several days) and consumes a significant amount of memory, interfering with query response times.

During the reindexing process, the database can not process updates.

When new data is added to the database, a full copy of the object is kept but attributes may be reordered according to the order of the attributes specified in the configuration file for each object type.

Multiple valued attributes (aatributes which can appear more than once in an object) are stored by concatenating the values of all the ocurrences separated by newline characters. This preserves attribute value order within an attribute type but transforms them into contiguous lines on output. For example, if the original had several non-contiguous remarks fields, their position inside the object is lost and only their relative order is maintained so that a user query will return all these values consecutively one after the other and not in their original positions.

In addition to the database itself, the current system stores some other pieces of information.

All incoming mails are stored in archive files in case they are needed in the future to track operating problems.

Every operation on a database object generates one (creation and deletion) or two files (modification) called serial files that have a unique identifier, a serial number, automatically assigned by the database. These are used to support the *near real time mirror* feature of the database.

2 The new storage system: Goals and constraints

2.1 Introduction

This section gives an overview of the main goals and constraints that come into the design of the new Database implementation.

When re-implementing the Database we want to keep the good features of the current system as much as possible while improving it by supporting the new needed features.

2.2 RPSL

The new Database will be the first RIPE Database to have support for RPSL [1].

RPSL has certain features that impose restrictions to storage. These affect not only the routing registry objects but all the objects in the Database.

Some of these new features such as line continuation don't affect the way objects are stored significantly. However, some other features do.

Particularly:

- *Attribute order is relevant*: This means that when displaying the object on a query the attributes must appear on the same order as the user sent them to the database. This is very useful for example when using multiple *remarks* lines to comment on the values of the following or preceding attributes (see figure 1 for an example).
- *End of line comments*: RPSL offers the possibility of including comments on any attribute value. This is done by typing the # character followed by the desired comment. Any text found after the # character is considered a comment and not parsed for syntax. If attribute values are to be stored as they are input by the user, this would result in all object value types being alphanumeric fields in the database.

```
route: 193.0.0.0/23
remarks: This route is announced out of AS3333
origin: AS3333
remarks: It is maintained by the RIPE NCC
mnt_by: RIPE-NCC-MNT
...
```

Figure 1: Example RPSL object with multiple occurrences of an attribute

```
route: 193.0.0.0/23 # This a route for a /23
...
```

Figure 2: Example of end of line comment

Storing multiple valued attributes as a single value of a column for a particular Database row, like the current Database does, would imply storing position information at the beggining of each segment of information, leading to a not very clean design and increasing the amount of processing needed during object storage and retrieval.

Also end of line comments would have to be stored on separate columns or they would interfere with indexing.

The easiest and most efficient way to solve this problems is to store a full text copy of the object as it was originally sent to the database in a text blob field with no structure.

This can also provide a fast way of answering queries where the full object is requested.

2.3 Object versioning

One of the desired features of the new database is to have the ability to look at how objects looked in the past. This feature may or may not be generally accessible to final users due to privacy issues (this will be discussed elsewhere).

This is very useful when tracking some operational problems.

In addition to just being able to retrieve an object's history we want the new database to provide the capability of retrieving the update (eg. e-mail) that created a particular version of an object.

This is needed to support transaction/registration replays that may be needed for some security functionality (eg. rps-auth [6]). It may also be used to provide audit trails.

As stated before, the current RIPE Database system stores all incoming e-mail updates in log files. However there is currently no fast and easy way to retrieve these when looking for some specific piece of data.

Keeping object history and making it easily accessible, even if only for the database operator, may have some privacy implications. See the proposed implementation below on how to deal with these implications while still providing this capability.

This feature should be optional and configurable by the DB administrator.

2.4 Speed

One of the reasons for re-implementing the RIPE Database is the desire to have a faster system.

From this follows that the database schema must provide an adequate mapping of the objects to the Database engine of choice allowing it to provide fast responses.

For some types of queries no disk-based system will provide enough speed and an in-memory data structure will be necessary to give adequate response times.

2.5 Robustness

The design should ensure no data is lost in the event of program or machine crashes.

The mechanism for carrying out updates without loss of data, taking into account the current schema and the capabilities of the supported RDBMS's will be discussed in the description of the update process [5].

2.6 Storage efficiency

Naturally the new design should not require an excessive amount of storage space that would make it impractical to operate.

Currently this is the least important of the requirements to keep in mind due to the availability of cheap, big storage devices. However it should be kept in mind.

2.7 Continuous operation

The new system should aim to minimize , and preferably eliminate, periods of service unavailability such as the ones present when the current system is going through data cleanup and reindexing.

Eventually, having a system capable of operating while performing the necessary administrative tasks (such as backup) would be ideal.

Also desirable is a mechanism to ensure smooth recovery from system crashes or other unexpected downtime.

2.8 Summary

As a summary these are the desired features that the new Database schema should have:

- keep an image of the object as it was submitted (RPSL req.)
- support object versioning
- store the message sent by the user
- be as fast as possible

- use storage as efficiently as possible while providing adequate speed
- be able to provide uninterrupted operation
- provide mechanisms to ensure a smooth recovery in the event of an unexpected system shutdown and avoid data loss.

3 The new storage system: Design choices

3.1 Choice of Database backend

When looking for an adequate backend to provide the Database storage and lookup functionality needed there are certain desirable features.

Ideally it should

- be as fast as possible
- be able to maintain performance on increased demand (number of database operations and size of data set)
- implement or, at least, make it easy to implement, transaction/rollback-like mechanisms
- be reliable
- behave as a black box from the programmers point of view
- have adequate support, and as much in-house knowledge as possible
- have free implementations so the software can be used without the need of purchasing commercial products

The main two choices for database backends fall into two categories:

- SQL RDBMSs
- Berkeley DB packages

The latest versions of the Berkeley database package provide support for fast storage of data in hash key/value pairs. They are free, relatively fast and lightweight.

However, we would like more flexibility than just key/value pairs in the hashes and B-trees and simple searches supported by the Berkeley DB [2].

For SQL RDBMSs there are both freely available and commercial implementations.

Among the freely available implementations the two most popular and best supported candidates are postgreSQL [3] and mySQL [4].

There are several commercial implementations of RDBMSs that support SQL, usually in a more complete way than the freely available ones. Some allow backups to be performed while the database is being accessed and have support for huge datasets while providing good performance.

Supporting SQL databases should give a more structured design for data storage and will need less changes if moving to one of the commercial RDBMS engines.

Therefore for the new database the backend of choice will be a SQL implementation. To start with, there will be support for one of the free implementations and after we have more experience with operating this and if we see significant additional features in the commercial implementations, one will be chosen and supported.

To be able to use several different database engines, a common API will be defined to avoid modifications to all upper layers of code.

The specific implementation of the API will be different for each database engine supported.

Please refer to the appropriate document for the API definition.

4 Draft Schema

This section describes the proposed schema for mapping the RIPE Database into a SQL databse following the relational database model.

The mapping of RIPE objects to SQL rows can be found in Appendix B.

4.1 Generic description of an object's representation

Object representation in the new database is radically different from what the current code does.

In general, objects will not contain *source* attributes but rather different sources will be stored in different SQL databases named according to the source value (eg: RIPE, APNIC, RADB, etc.).

For each object type there is a corresponding table in the SQL Database. However, not all information about an object is stored in that table. Representing many-to-many relationships is impossible to do cleanly on a SQL database using only one table. The normal solution is to use auxiliary tables that describe the relationship. These tables are defined in the next section.

Attributes on which no search will be carried out (they are not lookup or inverse keys or the information stored in them does not need to be accessed quickly) don't need to be stored in the SQL table for the object. In the current schema we use the mechanism used for object versioning for doing this.

4.2 Resolving many-to-many relationships

Many-to-many relationships refers to the situation where an object of a certain type is linked to many other objects (of possibly different types) and where, in turn, those objects are related to possibly more than one of the previous objects.

In the RIPE Database that kind of situation arises in several places. As an example, we will take the tech_c attribute.

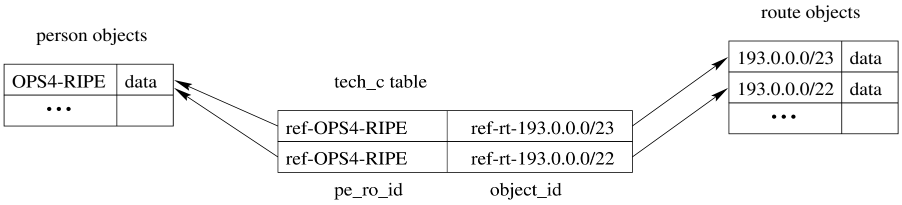

Figure 3: Example of auxiliary table

For example, an inetnum object can have multiple tech_c attributes pointing to different person or role objects. These person or role objects may, in turn, be referenced by more than one inetnum object or other type of object.

The best representation for this kind of relationship is an auxiliary table consisting of rows whose two columns are references to the related objects. Since a value can appear more than once in any of the two columns on any row, the primary key for these tables is the combination of the two columns (see figure 3).

4.3 Attributes stored in explicit SQL columns

Searchable attributes other then the object's primary keys and the attributes represented by auxiliary tables need to be stored in the main table for that object and indexed by the SQL engine.

There is no need to store end of line comments or whitespace since the object will be displayed from the full text copy.

4.4 Attributes not stored in explicit SQL columns

These attributes are stored only in the full text representation of the object. They are not searchable and are just static pieces of information.

4.5 General layout of an object

All objects can be seen as having two logical parts, as seen in Figure 4.

Serial # (ID)	DATA

Figure 4: A generic object

The **Serial #** part is a unique number assigned to the object by the system when it is created. It is also the primary key for every object in every table. It is unique within a particular Database (and therefore within any of the tables in that database).

Data refers to whatever data needs to be stored for a particular object type.

From a user's point of view, the primary keys will still be the ones defined as such in the whois templates.

From the database point of view these columns will be bound together into a unique key.

4.6 Storage required

Compared to the current system, the new Database implementation will require some more storage because of the different way of storing data and the additional requirements.

Mirror sites will see an increase in storage needs. We have considered this drawback and decided that the design simplicity and speed gain more than compensated for this, especially taking into account the decreasing price of hard disks.

4.7 Implementing object versioning

Object versioning will be implemented by creating some extra tables with a rather different structure than what has been discussed so far.

In general, rows in these tables will look like figure 5

Serial #	Text	Previous Serial #

Figure 5: Layout of rows for history tables

where

Serial # the primary key for the table. It is the same number defined in the previous section.

Text refers to the text representation of the object as submitted by the user, including white space, end of line comments and conserving the attribute order. It is an exact copy of the object as sent by the user.

Previous Serial # is a link to the previous version of the object.

There will be several tables using this format. Each will contain the objects for a particular interval of time. This way old copies can be stored on secondary storage devices, possibly off-site.

During normal operation the database will have two of these tables open.

One, which we will call *last*, will store the current versions of the objects.

The other contains previous versions of the objects and is also filled whenever an update is processed.

This second table is closed periodically, stored somewhere else and a new, empty table is opened. This should enable the Database operators to deal with privacy concerns regarding easy access to past (or even deleted) information in the Database by having only recent data easily accessible. Older data would be offline and would only be retrieved by fetching the offline files when a "good" reason is raised.

Therefore, this second table will contain a record of all updates processed by the database during a certain period of time.

In order to facilitate information retrieval in these tables, the system will ensure that all registrations corresponding to all transactions between the first serial number and the last one stored in the table are in that table.

This will avoid the creation of offline indexes for the contents of each history table and make it easy to know which files contain the elements of the object's history.

4.8 Object relationship

Figures 6,7 and 8 show how different objects relate to each other. In the diagrams, objects are represented by named rectangles. Auxiliary tables used to link objects are indicated by the names of the attributes they represent.

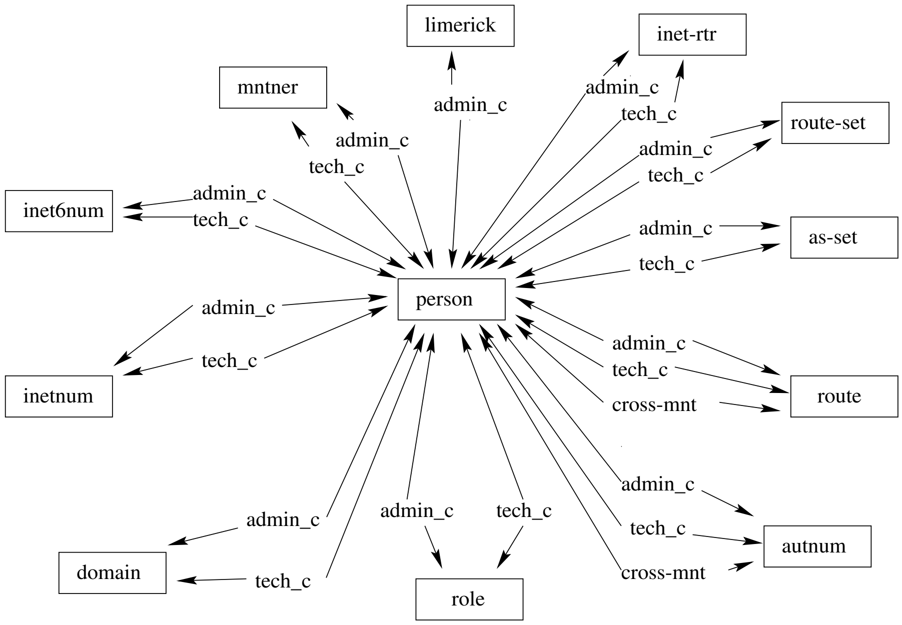

Figure 6: Relationships to person/role objects

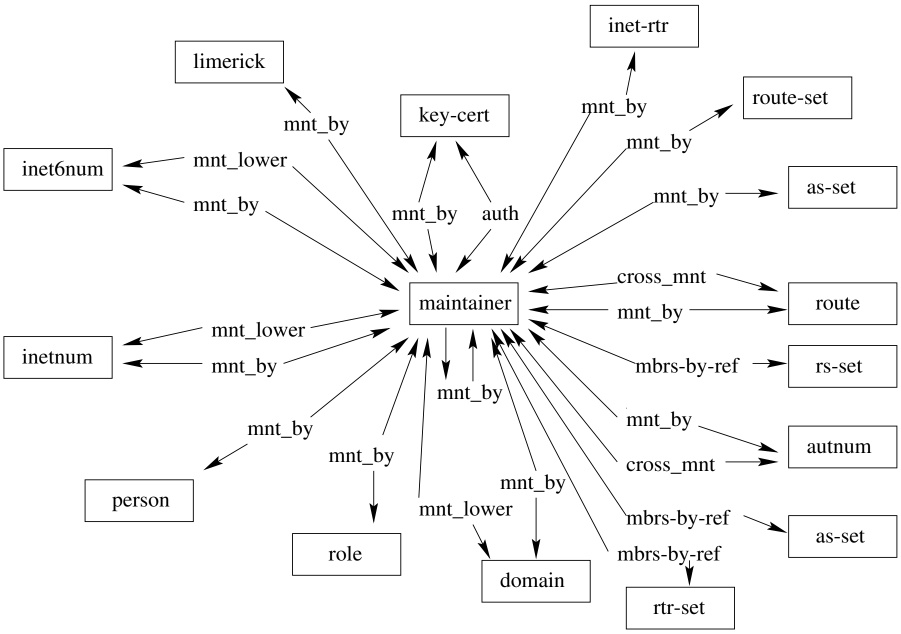

Figure 7: Relationships to maintainer objects

A Object templates

Whois object templates for reference

A.1 Objects common to all registries

A.1.1 Person and role objects

```
person:      [mandatory]  [single]     [primary/look-up key]
address:     [mandatory]  [multiple]
phone:       [mandatory]  [multiple]
fax-no:      [optional]   [multiple]
e-mail:      [optional]   [multiple]   [look-up key]
nic-hdl:     [mandatory]  [single]     [primary/look-up key]
remarks:     [optional]   [multiple]
notify:      [optional]   [multiple]   [inverse key]
mnt-by:      [optional]   [multiple]   [inverse key]
changed:     [mandatory]  [multiple]
source:      [mandatory]  [single]

role:        [mandatory]  [single]     [primary/look-up key]
address:     [mandatory]  [multiple]
phone:       [optional]   [multiple]
fax-no:      [optional]   [multiple]
e-mail:      [mandatory]  [multiple]   [look-up key]
```

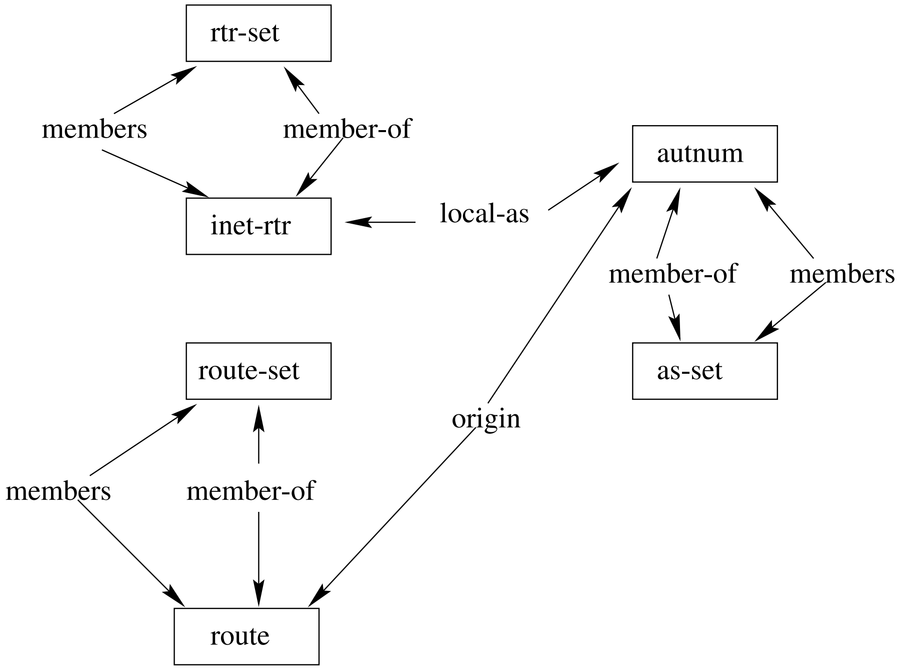

Figure 8: Other relationships

```
trouble:      [optional]   [multiple]
admin-c:      [mandatory]  [multiple]   [inverse key]
tech-c:       [mandatory]  [multiple]   [inverse key]
nic-hdl:      [mandatory]  [single]     [primary/look-up key]
remarks:      [optional]   [multiple]
notify:       [optional]   [multiple]   [inverse key]
mnt-by:       [optional]   [multiple]   [inverse key]
changed:      [mandatory]  [multiple]
source:       [mandatory]  [single]
```

A.1.2 Maintainer object

```
mntner:       [mandatory]  [single]     [primary/look-up key]
descr:        [mandatory]  [multiple]
admin-c:      [mandatory]  [multiple]   [inverse key]
tech-c:       [optional]   [multiple]   [inverse key]
upd-to:       [mandatory]  [multiple]
mnt-nfy:      [optional]   [multiple]
auth:         [mandatory]  [multiple]
remarks:      [optional]   [multiple]
notify:       [optional]   [multiple]   [inverse key]
mnt-by:       [mandatory]  [multiple]   [inverse key]
changed:      [mandatory]  [multiple]
source:       [mandatory]  [single]
```

A.1.3 Key cert object

```
key-cert:     [mandatory]  [single]     [primary/look-up key]
method:       [generated]  [single]
owner:        [generated]  [multiple]
fingerpr:     [generated]  [single]
certif:       [mandatory]  [multiple]
remarks:      [optional]   [multiple]
notify:       [optional]   [multiple]   [inverse key]
mnt-by:       [mandatory]  [multiple]   [inverse key]
changed:      [mandatory]  [single]
source:       [mandatory]  [single]
```

A.2 IP registry

A.2.1 IPv4 registration object

```
inetnum:      [mandatory]  [single]     [primary/look-up key]
netname:      [mandatory]  [single]     [look-up key]
descr:        [mandatory]  [multiple]
country:      [mandatory]  [multiple]
admin-c:      [mandatory]  [multiple]   [inverse key]
tech-c:       [mandatory]  [multiple]   [inverse key]
rev-srv:      [optional]   [multiple]   [inverse key]
status:       [mandatory]  [single]
remarks:      [optional]   [multiple]
notify:       [optional]   [multiple]   [inverse key]
mnt-by:       [optional]   [multiple]   [inverse key]
mnt-lower:    [optional]   [multiple]
changed:      [mandatory]  [multiple]
source:       [mandatory]  [single]
```

A.2.2 IPv6 registration object

```
inet6num:     [mandatory]  [single]     [primary/look-up key]
netname:      [mandatory]  [single]     [look-up key]
descr:        [mandatory]  [multiple]
country:      [mandatory]  [multiple]
admin-c:      [mandatory]  [multiple]   [inverse key]
tech-c:       [mandatory]  [multiple]   [inverse key]
rev-srv:      [optional]   [multiple]
status:       [mandatory]  [single]
remarks:      [optional]   [multiple]
notify:       [optional]   [multiple]   [inverse key]
mnt-by:       [optional]   [multiple]   [inverse key]
```

```
mnt-lower:   [optional]   [multiple]
changed:     [mandatory]  [multiple]
source:      [mandatory]  [single]
```

A.3 Routing registry. RPSL version

Note: this includes the RIPE extensions cross-mnt and cross-nfy.

A.3.1 Route object

```
route:          [mandatory]  [single]    [primary/look-up key]
descr:          [mandatory]  [multiple]
origin:         [mandatory]  [single]    [primary/inverse key]
holes:          [optional]   [single]
member-of:      [optional]   [multiple]  [inverse key]
inject:         [optional]   [multiple]
aggr-mtd:       [optional]   [single]
aggr-bndry:     [optional]   [single]
export-comps:   [optional]   [single]
components:     [optional]   [single]
remarks:        [optional]   [multiple]
admin-c:        [mandatory]  [multiple]  [inverse key]
tech-c:         [mandatory]  [multiple]  [inverse key]
cross-mnt:      [optional]   [multiple]  [inverse key]
cross-nfy:      [optional]   [multiple]  [inverse key]
notify:         [optional]   [multiple]  [inverse key]
mnt-by:         [mandatory]  [multiple]  [inverse key]
changed:        [mandatory]  [multiple]
source:         [mandatory]  [single]
```

A.3.2 Autonomous system object

```
aut-num:        [mandatory]  [single]    [primary/look-up key]
as-name:        [optional]   [single]
descr:          [mandatory]  [multiple]
member-of:      [optional]   [multiple]  [inverse key]
import:         [optional]   [multiple]
export:         [optional]   [multiple]
default:        [optional]   [multiple]
remarks:        [optional]   [multiple]
admin-c:        [mandatory]  [multiple]  [inverse key]
tech-c:         [mandatory]  [multiple]  [inverse key]
cross-mnt:      [optional]   [multiple]  [inverse key]
cross-nfy:      [optional]   [multiple]  [inverse key]
notify:         [optional]   [multiple]  [inverse key]
```

```
mnt-by:        [mandatory]  [multiple]  [inverse key]
changed:       [mandatory]  [multiple]
source:        [mandatory]  [single]
```

A.3.3 Internet router object

```
inet-rtr:      [mandatory]  [single]    [primary/look-up key]
alias:         [optional]   [multiple]
local-as:      [mandatory]  [single]    [inverse key]
ifaddr:        [mandatory]  [multiple]  [look-up key]
peer:          [optional]   [multiple]  [inverse key]
member-of:     [optional]   [multiple]  [inverse key]
remarks:       [optional]   [multiple]
admin-c:       [mandatory]  [multiple]  [inverse key]
tech-c:        [mandatory]  [multiple]  [inverse key]
notify:        [optional]   [multiple]  [inverse key]
mnt-by:        [mandatory]  [multiple]  [inverse key]
changed:       [mandatory]  [multiple]
source:        [mandatory]  [single]
```

A.3.4 Autonomous system set object

```
as-set:        [mandatory]  [single]    [primary/look-up key]
descr:         [mandatory]  [multiple]
members:       [optional]   [multiple]
mbrs-by-ref:   [optional]   [multiple]
remarks:       [optional]   [multiple]
tech-c:        [mandatory]  [multiple]  [inverse key]
admin-c:       [mandatory]  [multiple]  [inverse key]
notify:        [optional]   [multiple]  [inverse key]
mnt-by:        [mandatory]  [multiple]  [inverse key]
changed:       [mandatory]  [multiple]
source:        [mandatory]  [single]
```

A.3.5 Route set object

```
route-set:     [mandatory]  [single]    [primary/look-up key]
descr:         [mandatory]  [multiple]
members:       [optional]   [multiple]
mbrs-by-ref:   [optional]   [multiple]
remarks:       [optional]   [multiple]
tech-c:        [mandatory]  [multiple]  [inverse key]
admin-c:       [mandatory]  [multiple]  [inverse key]
notify:        [optional]   [multiple]  [inverse key]
mnt-by:        [mandatory]  [multiple]  [inverse key]
```

```
changed:        [mandatory]  [multiple]
source:         [mandatory]  [single]
```

A.3.6 Router set object

```
rtr-set:        [mandatory]  [single]    [primary/look-up key]
descr:          [mandatory]  [multiple]
members:        [optional]   [multiple]
mbrs-by-ref:    [optional]   [multiple]
remarks:        [optional]   [multiple]
tech-c:         [mandatory]  [multiple]  [inverse key]
admin-c:        [mandatory]  [multiple]  [inverse key]
notify:         [optional]   [multiple]  [inverse key]
mnt-by:         [mandatory]  [multiple]  [inverse key]
changed:        [mandatory]  [multiple]
source:         [mandatory]  [single]
```

A.3.7 Filter set object

```
filter-set:     [mandatory]  [single]    [primary/look-up key]
descr:          [mandatory]  [multiple]
filter:         [mandatory]  [single]
remarks:        [optional]   [multiple]
tech-c:         [mandatory]  [multiple]  [inverse key]
admin-c:        [mandatory]  [multiple]  [inverse key]
notify:         [optional]   [multiple]  [inverse key]
mnt-by:         [mandatory]  [multiple]  [inverse key]
changed:        [mandatory]  [multiple]
source:         [mandatory]  [single]
```

A.3.8 Peering set object

```
peering-set:    [mandatory]  [single]    [primary/look-up key]
descr:          [mandatory]  [multiple]
peering:        [mandatory]  [multiple]
remarks:        [optional]   [multiple]
tech-c:         [mandatory]  [multiple]  [inverse key]
admin-c:        [mandatory]  [multiple]  [inverse key]
notify:         [optional]   [multiple]  [inverse key]
mnt-by:         [mandatory]  [multiple]  [inverse key]
changed:        [mandatory]  [multiple]
source:         [mandatory]  [single]
```

A.3.9 Dictionary object

```
dictionary:    [mandatory]  [single]    [primary/look-up key]
descr:         [optional]   [multiple]
admin-c:       [mandatory]  [multiple]  [inverse key]
typedef:       [optional]   [multiple]
rp-attribute:  [optional]   [multiple]
protocol:      [optional]   [multiple]
remarks:       [optional]   [multiple]
notify:        [optional]   [multiple]  [inverse key]
mnt-by:        [mandatory]  [multiple]  [inverse key]
changed:       [mandatory]  [multiple]
source:        [mandatory]  [single]
```

A.4 Domain name registry

A.4.1 Domain object

```
domain:      [mandatory]  [single]    [primary/look-up key]
descr:       [mandatory]  [multiple]
admin-c:     [mandatory]  [multiple]  [inverse key]
tech-c:      [mandatory]  [multiple]  [inverse key]
zone-c:      [mandatory]  [multiple]  [inverse key]
nserver:     [optional]   [multiple]  [inverse key]
sub-dom:     [optional]   [multiple]  [inverse key]
dom-net:     [optional]   [multiple]
remarks:     [optional]   [multiple]
notify:      [optional]   [multiple]  [inverse key]
mnt-by:      [optional]   [multiple]  [inverse key]
mnt-lower:   [optional]   [multiple]
refer:       [optional]   [single]
changed:     [mandatory]  [multiple]
source:      [mandatory]  [single]
```

A.5 Secret registry

A.5.1 Limerick object

```
limerick:    [mandatory]  [single]    [primary/look-up key]
descr:       [optional]   [multiple]
text:        [mandatory]  [multiple]
admin-c:     [mandatory]  [multiple]
author:      [mandatory]  [multiple]  [inverse key]
remarks:     [optional]   [multiple]
notify:      [optional]   [multiple]  [inverse key]
```

```
mnt-by:       [mandatory]  [multiple]   [inverse key]
changed:      [mandatory]  [multiple]
source:       [mandatory]  [single]
```

B SQL representation

B.1 Objects common to all registries

```
#
# Table structure for table 'person_role'
#
CREATE TABLE person_role (
  pe_ro_id int(10) default '0' not null,
  is_person enum('p','r') default 'p' not null,
  nic_hdl varchar(20) DEFAULT '' NOT NULL,
  PRIMARY KEY (pe_ro_id,nic_hdl),
  index (nic_hdl(8))
);

#
# Table to store the names so we can do quick searches on individual
# words
#

CREATE TABLE names (
  pe_ro_id int(10) default '0' not null,
  name varchar(64) default '' not null,
  primary key(pe_ro_id,name),
  index (name(5))
);

#
# Table structure for table 'mntner'
#
CREATE TABLE mntner (
  mnt_id int(10) default '0' not null,
  mntner varchar(80) DEFAULT '' NOT NULL,
  PRIMARY KEY (mnt_id),
  index mntner (mntner(5))
);

#
# Table structure for table 'mnt_mnt_notify'
#
CREATE TABLE mnt_mnt_notify (
```

```
  mnt_id int(10) default '0' not null,
  mnt_nfy varchar(128) default '' not null,
  PRIMARY KEY (mnt_id,mnt_nfy),
  index (mnt_nfy(8))
);

#
# Table structure for table 'mnt_upd_to'
#
CREATE TABLE mnt_upd_to (
  mnt_id int(10) default '0' not null,
  upd_to varchar(128) default '' not null,
  PRIMARY KEY (mnt_id,upd_to),
  index (upd_to(8))
);

#
# Table structure for table 'mnt_auth'
#
CREATE TABLE mnt_auth (
  mnt_id int(10) unsigned DEFAULT '0' NOT NULL,
  auth varchar(244) DEFAULT '' NOT NULL,
  PRIMARY KEY (mnt_id, auth)
);

#
# Table structure for table 'key_cert'
#
CREATE TABLE keycert (
  key_cert_id int(10) default '0' not null,
  key_cert varchar(32) default '' not null,
  primary key (key_cert_id,key_cert),
  index (key_cert(5))
);

#
# tables common to all objects
#

#
# Table structure for table 'admin_c'
#
CREATE TABLE admin_c (
  object_id int(10) default '0' not null,
  pe_ro_id int(10) unsigned DEFAULT '0' NOT NULL,
  object_type tinyint unsigned default '0' not null,
```

```
  PRIMARY KEY (pe_ro_id,object_id),
  index (object_type)
);

#
# Table structure for table 'tech_c'
#
CREATE TABLE tech_c (
  object_id int(10) default '0' not null,
  pe_ro_id int(10) unsigned DEFAULT '0' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (pe_ro_id,object_id),
  index (object_type)
);

#
# Table structure for table 'zone_c'
#
CREATE TABLE zone_c (
  object_id int(10) default '0' not null,
  pe_ro_id int(10) unsigned DEFAULT '0' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (pe_ro_id,object_id),
  index (object_type)
);

#
# Table structure for table 'mnt_by'
#
CREATE TABLE mnt_by (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  mntner varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, mntner),
  index (mntner(5)),
  index (object_type)
);

#
# Table structure for table 'mnt_lower'
#
CREATE TABLE mnt_lower (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  mntner varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, mntner),
```

```
  index (mntner(5)),
  index (object_type)
);

#
# Table structure for table 'e_mail'
#
CREATE TABLE e_mail (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  e_mail varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, e_mail),
  index (e_mail(8)),
  index (object_type)
);

#
# Table structure for table 'notify'
#
CREATE TABLE notify (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  notify varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, notify),
  index (notify(8)),
  index (object_type)
);
```

B.2 IP registry

```
#
# Table structure for table 'inetnum'
#
CREATE TABLE inetnum (
  in_id int(10) unsigned default '0' NOT NULL auto_increment,
  begin_in int unsigned DEFAULT '0' NOT NULL,
  end_in int unsigned DEFAULT '0' NOT NULL,
  netname varchar(80) DEFAULT '' NOT NULL,
  PRIMARY KEY (in_id),
  KEY netname (netname)
);

#
# Table structure for table 'in_rev_srv'
#
CREATE TABLE in_rev_srv (
```

```
  in_id int(10) unsigned DEFAULT '0' NOT NULL,
  rev_srv varchar(244) DEFAULT '' NOT NULL,
  PRIMARY KEY (in_id, rev_srv)
);

#
# Table structure for table 'inet6num'
#
CREATE TABLE inet6num (
  i6_id int(10) unsigned default '0' NOT NULL auto_increment,
  begin_i6_msb int8 unsigned DEFAULT '0' not null,
  begin_i6_lsb int8 unsigned DEFAULT '0' not null,
  end_i6_msb int8 unsigned DEFAULT '0' not null,
  end_i6_lsb int8 unsigned DEFAULT '0' not null,
  netname varchar(80) DEFAULT '' NOT NULL,
  PRIMARY KEY (i6_id),
  KEY netname (netname)
);

#
# Table structure for table 'i6_rev_srv'
#
CREATE TABLE i6_rev_srv (
  i6_id int(10) unsigned DEFAULT '0' NOT NULL,
  rev_srv varchar(244) DEFAULT '0' NOT NULL,
  PRIMARY KEY (i6_id, rev_srv)
);
```

B.3 Routing registry. RPSL version

```
#
# Table structure for table 'route'
#
CREATE TABLE route (
  rt_id int(10) default '0' not null,
  route varchar(19) DEFAULT '' NOT NULL,
  origin varchar(10) DEFAULT '' NOT NULL,
  unique (route,origin),
  PRIMARY KEY (rt_id)
);

#
# Table structure for table 'member_of'
#
CREATE TABLE member_of (
  object_id int(10) unsigned DEFAULT '0' NOT NULL ,
```

```
  set_id int(10) unsigned DEFAULT '0' NOT NULL ,
  object_type tinyint unsigned default '0' not null,
  index (object_type),
  PRIMARY KEY (object_id,set_id)
);

#
# Table structure for table 'cross_mnt'
#
CREATE TABLE cross_mnt (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  mntner varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, mntner),
  index (mntner(5)),
  index (object_type)
);

#
# Table structure for table 'cross_nfy'
#
CREATE TABLE cross_nfy (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  notify varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, notify),
  index (notify(8)),
  index (object_type)
);

#
# Table structure for table 'aut_num'
#

CREATE TABLE aut_num (
  an_id int(10) default '0' not null,
  aut_num varchar(7) DEFAULT '' NOT NULL,
  KEY aut_num (aut_num),
  PRIMARY KEY (an_id)
);

#
# Table structure for table 'inet_rtr'
#
CREATE TABLE inet_rtr (
  ir_id int(10) unsigned DEFAULT '0' NOT NULL,
```

```
  inet_rtr varchar(254) DEFAULT '' NOT NULL,
  local_as varchar(7) DEFAULT '' NOT NULL,
  PRIMARY KEY (ir_id),
  unique (inet_rtr),
  KEY local_as (local_as)
);

#
# Table structure for table 'ifaddr'
#
CREATE TABLE ifaddr (
  ir_id int(10) unsigned DEFAULT '0' NOT NULL,
  ifaddr varchar(40) DEFAULT '' NOT NULL,
  PRIMARY KEY (ir_id, ifaddr)
);

#
# Table structure for table 'peer'
#
CREATE TABLE peer (
  ir_id int(10) unsigned DEFAULT '0' NOT NULL,
  peer varchar(40) DEFAULT '' NOT NULL,
  PRIMARY KEY (ir_id, peer)
);


#
# Table structure for table 'as-set'
#
CREATE TABLE as_set (
  as_id int(10) unsigned DEFAULT '0' NOT NULL ,
  as_set varchar(80) DEFAULT '' NOT NULL,
  PRIMARY KEY (as_id,as_set)
);

#
# Table structure for table 'mbrs_by_ref'
#
CREATE TABLE mbrs_by_ref (
  object_id int(10) unsigned DEFAULT '0' NOT NULL,
  mntner varchar(80) DEFAULT '' NOT NULL,
  object_type tinyint unsigned default '0' not null,
  PRIMARY KEY (object_id, mntner),
  index (mntner(5)),
  index (object_type)
);
```

```
#
# Table structure for table 'route-set'
#
CREATE TABLE route_set (
  rs_id int(10) unsigned DEFAULT '0' NOT NULL,
  route_set varchar(80) DEFAULT '' NOT NULL,
  KEY route_set (route_set),
  PRIMARY KEY (rs_id)
);

#
# Table structure for table 'filter_set'
#
CREATE TABLE filter_set (
  fs_id int(10) unsigned DEFAULT '0' NOT NULL,
  filter_set varchar(80) DEFAULT '' NOT NULL,
  filter longblob default '' not null,
  INDEX filter_set (filter_set),
  PRIMARY KEY (fs_id)
);

#
# Table structure for table 'peering_set'
#
CREATE TABLE peering_set (
  ps_id int(10) unsigned DEFAULT '0' NOT NULL,
  peering_set varchar(80) DEFAULT '' NOT NULL,
  PRIMARY KEY (ps_id)
);

#
# Table structure for table 'peering'
#
CREATE TABLE peering (
  pr_id int(10) unsigned DEFAULT '0' NOT NULL,
  peering longblob default '' not null,
  PRIMARY KEY (pr_id)
);
```

B.4 Domain name registry

```
CREATE TABLE domain (
  dn_id int(10) unsigned DEFAULT '0' NOT NULL,
  domain varchar(254) DEFAULT '' NOT NULL,
# this part for referral attr
```

```
  type varchar(244) DEFAULT 'RIPE' NOT NULL,
  port int(5) default '43' not null,
  host varchar(80) default '' not null,
  index domain (domain(5)),
  PRIMARY KEY (dn_id)
);

#
# Table structure for table 'dn_dom_net'
#
CREATE TABLE dn_dom_net (
  dn_id int(10) unsigned DEFAULT '0' NOT NULL,
  network char(15) DEFAULT '' NOT NULL,
  PRIMARY KEY (dn_id, network)
);

#
# Table structure for table 'dn_nserver'
#
CREATE TABLE dn_nserver (
  dn_id int(10) unsigned DEFAULT '0' NOT NULL,
  host varchar(244) DEFAULT '' NOT NULL,
  PRIMARY KEY (dn_id, host)
);

#
# Table structure for table 'dn_sub_dom'
#
CREATE TABLE dn_sub_dom (
  dn_id int(10) unsigned DEFAULT '0' NOT NULL,
  domain varchar(244) DEFAULT '' NOT NULL,
  PRIMARY KEY (dn_id, domain),
  index (domain(5))
);
```

B.5 Secret registry

```
#
# Table structure for table 'limerick'
#
CREATE TABLE limerick (
  li_id int(10) unsigned default '0' NOT NULL,
  limerick varchar(80) DEFAULT '' NOT NULL,
  PRIMARY KEY (li_id),
  UNIQUE limerick (limerick)
```

```
);

#
# Table structure for table 'li_author'
#
CREATE TABLE li_author (
  li_id int(10) default '0' not null,
  pe_ro_id int(10) default '0' not null,
  PRIMARY KEY (li_id,pe_ro_id)
);
```

B.6 Object history

```
create table serials (
  serial int(10) default '0' not null,
  prev_serial int(10) default '0' not null,
  object longblob default '' not null,
  PRIMARY KEY (serial)
);

create table last (
  serial int(10) default '0' not null,
  prev_serial int(10) default '0' not null,
  object longblob default '' not null,
  PRIMARY KEY (serial)
);
```

References

[1] RFC 2280 and draft-ietf-rps-rpsl-v2-03.txt

[2] http://abyssinian.sleepycat.com/db/

[3] http://www.postgresql.org

[4] http://www.tcx.se

[5] Description of the update process. To be written.

[6] Internet draft. draft-ietf-rps-auth-03.txt