

Prologue

This is the documentation for `rldb`. `rldb` stands for Record Library Database, and it is designed to enable the easy maintenance of a database cataloging a collection of (predominantly classical, in the record-shop sense) music.

The program uses the [Postgres](#) relational database engine to maintain the database, assuring rock-solid reliability and data integrity. It uses [python](#) as a programming language, and the user-interface is coded using [python-qt](#) (a.k.a. PyQt), the python language bindings for the [Trolltech's Qt widget set](#). `rldb` is therefore in principle portable across many platforms (although it was developed and tested on [Debian GNU/Linux](#)).

Chapter 1 covers the installation of the program and the one-off creation of the records database. Chapter 2 contains the users' guide and tips for building a database. Chapter 3 documents the code and should be read by programmers intending to add features or extend the data structures.

The author of `rldb` is Nick Bailey <nick@bailey-family.org.uk>. This software is distributed under the [GNU Public Licence](#) (GPL). You should have obtained a copy of the licence along with the software. You are free to do whatever you like with this software subject to the conditions of the licence, including copy it, give it away, or even sell it, but please as a matter of courtesy, email the author to inform him of any changes or extensions you are making. The program is used to maintain and query a database of a modest record and CD collection of some three thousand or so entries, and while it is believed to be reliable, there is no warranty or guarantee of suitability for this purpose, or anything at all. Once again, please refer to the licence for further details.

If you like the program so much that you want to donate to it, please consider supporting the [Free Software Foundation](#), or the [Electronic Frontier Foundation](#), both of which protect my rights to create, and your right to use this software, against corporate interests who are trying hard to suppress such activity. If you feel particularly kindly disposed towards the author, you can always look him up on [Amazon UK](#) and send him a CD from his wish list!

Chapter 1

Installation and Setting-up the Database

Unpack the tarball in a directory of your choice, for example, `/usr/local`. It creates a subdirectory called `rldb` which contains the binary `rldb.py` and other scripts and utilities to assist in the installation and maintenance of the database. Make the application visible by linking it to a place in your path. For example, as root you might issue the following commands to make the program execute under the name `rldb`:

```
cd /usr/local/bin
ln -s ../rldb/rldb.py rldb
```

If you have not already done so, obtain and install the postgres relational database server and the client applications, particularly `createdb` and `psql`. Also obtain and install the python bindings for the qt widget set, `pyqt`. These are used by the `rldb` user interface program.

Create a database called 'records' make the appropriate tables by executing the appropriate SQL commands. The SQL you need are in the file `createdb.sql` in the distribution directory. `cd` into that directory and proceed as follows:

```
nick@haydn:/usr/local/rldb$ createdb records
CREATE DATABASE
nick@haydn:/usr/local/rldb$ psql records < createdb.sql
```

Many messages will appear of the form `NOTICE: CREATE TABLE will create implicit sequence...` and this is normal behaviour, but there should be no warnings or errors if everything is going to plan.

Run `rldb` to test the database and widget set installations. The application window shown in Figure 1.1 appears if installation was successful.

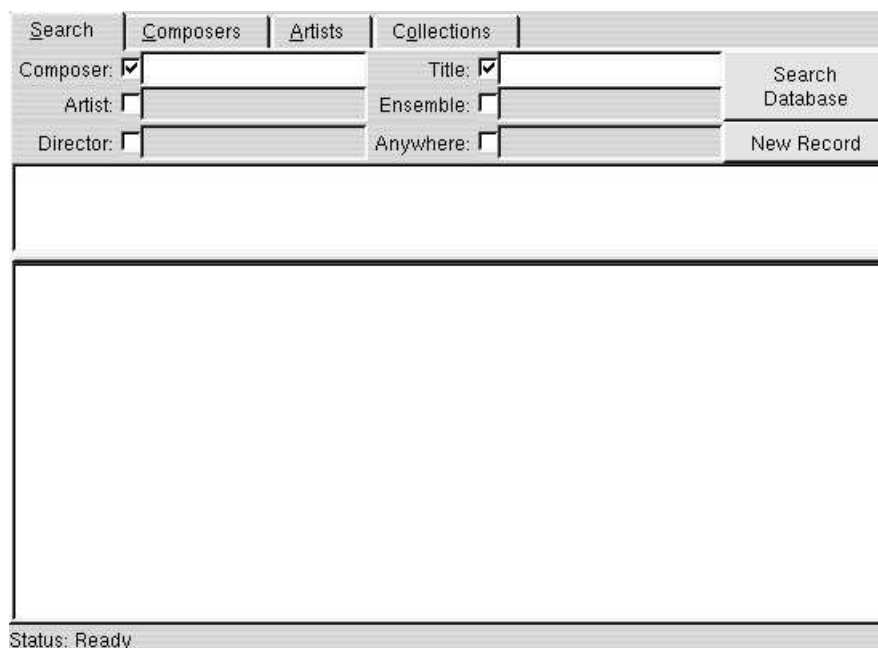


Figure 1.1. Start-up with an Empty Database

Search	Composers	Artists	Collections
Daniel Ciampolini/Xylophone & Glockenspiel			
Dimitri Vassilakis/Piano			
Ensemble InterContemporain			
Florent Boffard/Piano			
Frederique Cambreling/Harp			
Michel Cerutti/Cimbaloni			
Pierre Boulez			
Vincent Bauer/Vibraphone			
Name: Ensemble InterContemporain			Clear
Instrument:			Update
			Add New
			Refresh List
Status: Selected Record [8]			

Figure 2.2. The Artist Browser

Return to the new-record dialogue and fill in the fields. Free-format fields can be filled in directly. To set the composer, either click the Composer button on the dialogue to set the composer to the currently selected one in the composer browser, or equivalently, double click an entry in the composer browser window. Nominate artists in the same way; the list will be built up in the order you select them, but the order can be changed with the Move Up or Move Down buttons on the dialogue, or a selected artist removed with the Delete button. If necessary, the director and ensemble can be recorded by selecting the relevant line in the Artists browser and then clicking the associated button in the dialogue. When all the data have been entered, the dialogue will have an appearance something like that in Figure 2.3.

Composer:	Pierre Boulez	
Title:	Repons	
Opus	Key	Date
		1981-1984
Ensemble:	Ensemble InterContemporain	
Director:	Pierre Boulez	
Artists:	[1]: Dimitri Vassilakis, Piano	
Add	[2]: Florent Boffard, Piano	
Delete	[3]: Frederique Cambreling, Harp	
Move Up	[4]: Vincent Bauer, Vibraphone	
Move Down	[5]: Daniel Ciampolini, Xylophone & Glockenspiel	
	[6]: Michel Cerutti, Cimbaloni	
Location:		
Medium:	CD	
Comments:		
Commit Changes	Add as New Record	Delete this Record

Figure 2.3. The New Record Dialogue with Data Entered

Because the dialogue was invoked in order to add a new record, the only action button which is enabled is the Add New Record button. Clicking this now enters the record into the database. Once accepted, the database is instantly updated, and there is no need to perform any ‘save’ operation before closing down the program. Note that a new record must have a composer and a title, although these need not be unique.

Check that the record has been entered correctly by looking for it on the Search pane of the main application window. The short-form should be visible in the upper half, and the lower half should show the full card, as in Figure 2.4.

The screenshot shows a software window with a search form at the top and a record card below. The search form has tabs for 'Search', 'Composers', 'Artists', and 'Collections'. It contains several input fields with checkboxes: 'Composer: ', 'Artist: ', 'Director: ', 'Title: ', 'Ensemble: ', and 'Anywhere: '. There are two buttons: 'Search Database' and 'New Record'. Below the form, a search result is displayed in a blue bar: 'Pierre BOULEZ: Repons (1981-1984): Ensemble InterContemporain/Pierre Boulez'. The record card below shows the following details:

Pierre Boulez
Repons
(1981-1984)
Ensemble InterContemporain, Pierre Boulez [CD]
 Dimitri Vassilakis, Piano
 Florent Boffard, Piano
 Frederique Cambreling, Harp
 Vincent Bauer, Vibraphone
 Daniel Ciampolini, Xylophone & Glockenspiel
 Michel Cerutti, Cimbaleri

Status: New record [1] created

Figure 2.4. The Newly-entered Record as Displayed

2.2 Editing existing data

You can change the entry associated with a record later by double-clicking on the list of search hits. The order of artists can be changed using the move-up and move down buttons, more added or deleted, and other fields edited. Be sure to click the Commit Changes button rather than Add as New Record, otherwise you will generate a new database entry with the modified data rather than replacing the existing one.

Deletion of an entire record is accomplished by clicking on the Delete this Record button, but note that there is no automatic removal of artists, composers etc. from their associated lists just because they are no longer referred to in the database. In fact, rldb does not offer any method for the deletion of a composer or artist entry. There is nothing to stop you reusing an artist or composer entry by simply editing its fields, but beware that there isn't a record you've forgotten about somewhere which is referring to that artist or composer!

2.3 Performing Searches

rldb permits general searching across the whole of the database, or on the content of particular fields. The composer, title, artist, ensemble or director search criteria may be specified independently, or a search condition may be given for text appearing anywhere in the record. Where more than one condition is specified, they are combined conjunctively (which is to say that all of the conditions have to be satisfied). The search condition is only applied when the check-box is ticked

Composer: <input checked="" type="checkbox"/>		Title: <input checked="" type="checkbox"/>	
Artist: <input type="checkbox"/>		Ensemble: <input type="checkbox"/>	
Director: <input type="checkbox"/>		Anywhere: <input type="checkbox"/>	

Figure 2.5. Search Criteria

Search terms are case-insensitive, so Bach, BACH and bach all match the same text. With the exception of the Composer field, the search terms are posix regular expressions. To find an exact match, use the `^` and `$` (respectively, start- and end-of line); for example, `^concerto for flute$` does not match *Concerto for Flute and Violin*, whereas it would if the `$` was omitted. However, beware particularly restrictive regular expressions: this one won't match *Flute Concerto* with or without the line delimiters.

The Composer field is interpreted as one or two (comma-separated) regular expressions, the former matching the composer's surname and the latter the forenames. So `bach,j s|c p e` matches both of the Bach's, but neither W F nor J C. Once again, use with caution: neither JS Bach nor CPE Bach will match. One might try `bach,j ?s|c ?p ?e` instead (the question-mark indicating that the previous token, in this case the space, is optional), but this is getting rather over-complicated.

2.4 Compilations

Once in a while, you get hold of a CD which it's almost impossible place on your shelf, because the title contains no clue as to the composer or title of the contents in the spine. "Baroque Harpsichord Music by Women Composers", for example. Alternatively, you might have made a compilation of diverse bits and pieces for the car, and called it "Now That's What Nick Calls Music #22". The solution for this is to decide on an arbitrary place in your collection for the offending item, then use the Location tab to connect all of the pieces in your index which are now by accident rather than design stored in the same place.

Since I always use a composer/title order for my collection, the fields which this table supports are Filed with (Composer) and (Title). When you attach a location to a piece, the string *Filed under...* shows up on the card, as seen in Figure 2.6.

Search	Composers	Artists	Collections
Composer: <input type="checkbox"/>	<input type="text"/>	Title: <input type="checkbox"/>	<input type="text"/>
Artist: <input type="checkbox"/>	<input type="text"/>	Ensemble: <input type="checkbox"/>	<input type="text"/>
Director: <input type="checkbox"/>	<input type="text"/>	Anywhere: <input checked="" type="checkbox"/>	durham
			Search Database
			New Record
ALAIN: Postlude pour l'Office de Complies			
HOWELLS: Psalm-prelude, Op 23 No 3			
LANGLAIS: Hymne d'Action de Grace "Te Deum": James Lancelot/Durham Cathedral Organ			
<h2>Langlais</h2> <h3>Hymne d'Action de Grace "Te Deum"</h3> <p>James Lancelot, Durham Cathedral Organ <i>Filed under "Langlais: Durham Cathedral" [CD]</i></p>			
Status: 8 records matched			

Figure 2.6. A Work with an Associated Location

Chapter 3

Programmers' Guide

3.1 Structure of the Database

The package comes with a script, `createdb.sql`, which sets up the postgres database. The ERD for the database is shown in Figure 3.1. The primary table from the point of view of the python application is **Work**. The **Work** table references by foreign key the **Composer** table, **Location** table, and **Artist** table. With the exception of the primary and foreign key fields, all of the fields are of type text. This is required because even those which look as if they might be typed, for example **Composer.born**, might end up with values like `c. 1200` which even if one was prepared to represent as `01 Jan 1200` is very likely to be out-of-range of most relational databases because it is before the Unix Epoch.

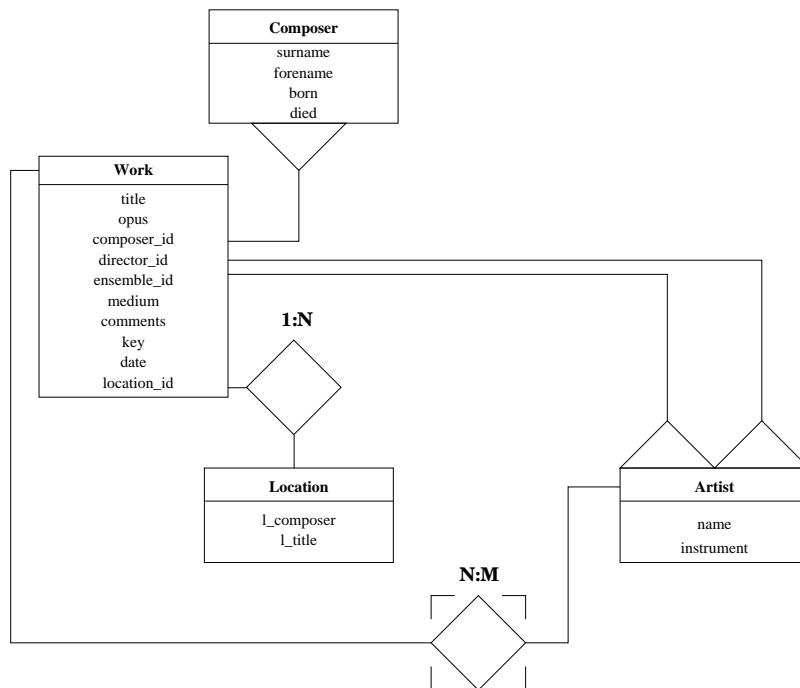


Figure 3.1. Entity Relationship Diagram for the Database

Points worthy of note:

- The field **Work.key** refers to the key of the piece; it is not the primary key!
- **Work.medium** is a convenience field for searches of the form “find works on CD”. It is not intended for storing the record number. That would best be stored under **Location**, but since the author is not inclined to type in record numbers, this is unimplemented. Users sufficiently fastidious to record all of the CD numbers they have in their collection are probably assiduous programmers, and are encouraged to add the feature if they desire.
- **Artist** is referenced in many ways. It is used to store a list of performers as a many-many relation. A table called **Work_artist** is used to implement the many-many relation between artists and works. This is implemented by an additional integer field in the record called **Work_artist.ord** which determines the display order of the artists associated with a particular work. **Artist** is used to store information about ensembles and directors. These are regarded as special cases of artists, with the instrument field empty.

3.2 The Python Script

The main application script, `rldb.py`, is a monolithic executable containing all of the code of the database application. While one could think of more elegant ways of organising the code, this has the advantage of not requiring any adjustment to search paths, library paths and so on, and makes unsuccessful installation really rather difficult.

The code is essentially broken into two halves. The first part is responsible for opening a

connection to Postgres, and creating well-formed SQL enquiries to pass to the relational database. The second half maintains a list of tuples containing matching records in a format internal to the application, and catches all of the user-interface generated events.

rldb imports the following:

sys	Misc operating system functions, only used to access command line argument list.
qt	Python binding for the QT widget set.
psycopg	Postgres access library, threadsafe alternative to the pg module.
re	Regular expressions in python.
copy	Deep copy for lists (like 'clone' in Java)

Table 3.1. Imported modules

3.2.1 Global Variables and Functions

Internal data is maintained by tuples of length `WorkMax`. The fields of the tuple can be indexed using the global variables defined at the beginning of the script: `Id`, `ComposerId`, `Surname`, and so on. The global variable `empty` is such a tuple with the fields preinitialised so that they have the correct type (i.e. mostly `'`, although some fields contain `0`)

Utility functions `locationStr` and `artistStr` format available data in human-readable form inserting punctuation as required.

3.2.2 Classes and Methods

Class descriptions here

3.2.3 Known Bugs

- When an existing record is changed, it can appear duplicated in the search hits list. Both entries have the same record ID, so there is no risk of database corruption, but this is visually annoying.

3.2.4 Possible Improvements

- For the sake of efficiency, much of the SQL query engine should be moved into the database itself by using persistent functions. This would mean adding the code into the `createdb.sql` script.
- There is no way of directly erasing the `Ensemble` or `Director` field. The work-around is to make a record in the artists table which has an ID of `0` and empty strings for its fields. Selecting this places the required values in the database records. But this is a nasty hack.
- A `Merge` function would be really useful so you can combine two artist entries to be the same. For example, you might inadvertently have separate entries for “Emma Kirkby” and “Emma Kirkby, Soprano”, or have mis-spelled the artist on one occasion and not noticed until after having created another record.
- A `Utilities` panel for housekeeping work. A database dumping facility would be very useful and encourage a responsible attitude to backups.