Demo 1: Introducing Pyrrho DBMS

Malcolm Crowe, 25 Oct 2022

[Slide 7 @ 03:47 in the Tutorial video]

Our first demonstration is about the transaction log. The transaction log defines the contents of the database. This tutorial also works with very few changes on Linux and MacOS.

If we think of a transaction commit as comprising a set of elementary operations e, then the transaction log is best implemented as a serialization of these events to the append storage. We can think of these serialized packets as objects in the physical database. In an object-oriented programming language, we naturally have a class of such objects, and we call this class Physical.

So, at the commit point of a transaction, we have a list of these objects, and the commit has two effects (a) appending them to the storage, (b) modifying the database so that other users can see.

We can think of each of these elementary operations as a transformation on the database, to be applied in the order they are written to the database (so the ordering within each transaction is preserved). And the transaction itself is the resulting transformation of the database.

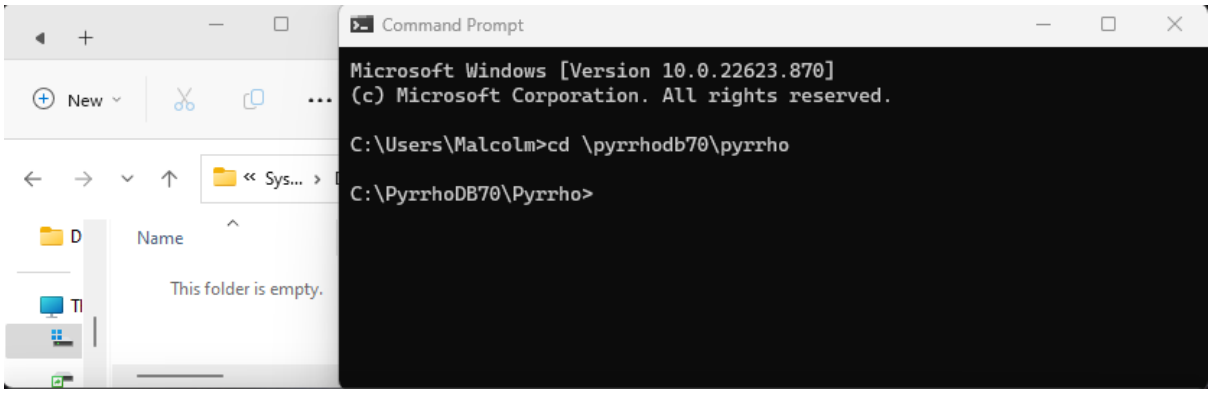
Any state of the database is thus the result of the entire sequence of committed transactions, starting from a known initial database state corresponding to an empty database.

[8 @ 05:06]

Let's start with a command window (or terminal), set to whatever folder the distribution is in. Don't worry if yours is different.

The Pyrrho distribution folder contains the server and command line processor executable files, and the PyrrhoLink dll. (You can copy these files to anywhere convenient.)

This command window will be for the server.

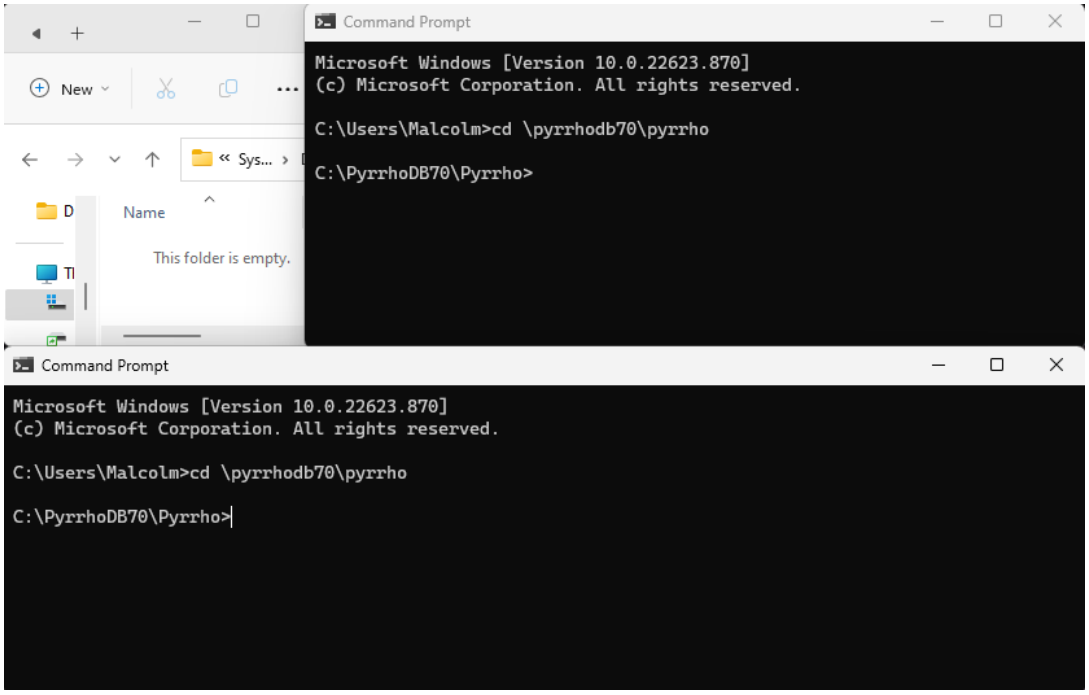


[9 @ 05:29]

Let us agree on a folder for the database files. Create a new folder \DATA. To save space, as here, you can overlap the windows a bit.

[10 @ 05:42]

Before we start the server, let's add a command window for the client, also with the same folder. We will see it is better to make it wider than normal.

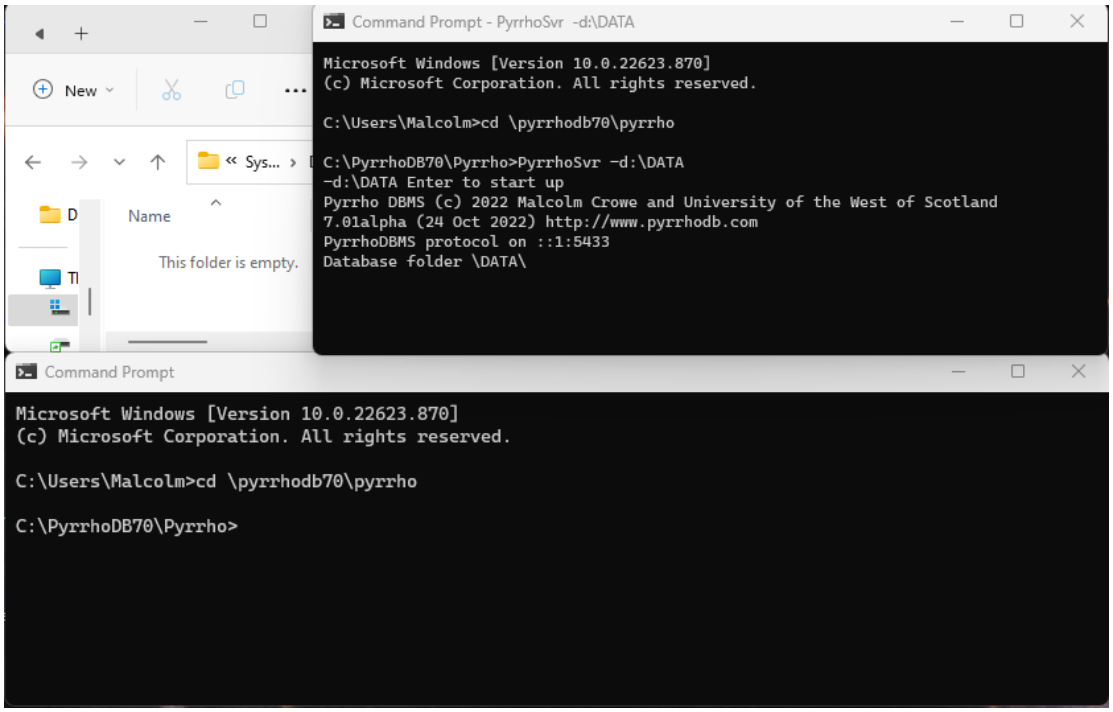


[11 @ 05:54]

The Pyrrho server is called PyrrhoSvr, and it runs in an ordinary account with no special privileges. You can copy it to anywhere convenient.

It is easiest just to use it in a command window. When we specify the server, we can provide a folder for it to store database files in. In the command window give the command (for Linux and MacOS you need to prefix this command with mono: **mono PyrrhoSvr.exe -d:/DATA**):

`PyrrhoSvr -d:\DATA`



[12 @ 06:16]

When prompted, we verify that the options specified are the ones we want and click Enter.

[13 @ 06:26]

You then need to leave the command window open (of course you can minimise it). It can be useful for diagnostic information if something goes wrong.

On startup, Pyrrho checks that it can access the specified folder, but places nothing in it. Databases are created by users, and the transaction log will be stored by the server on disk, usually in this default folder. It also announces its TCP/IP request port.

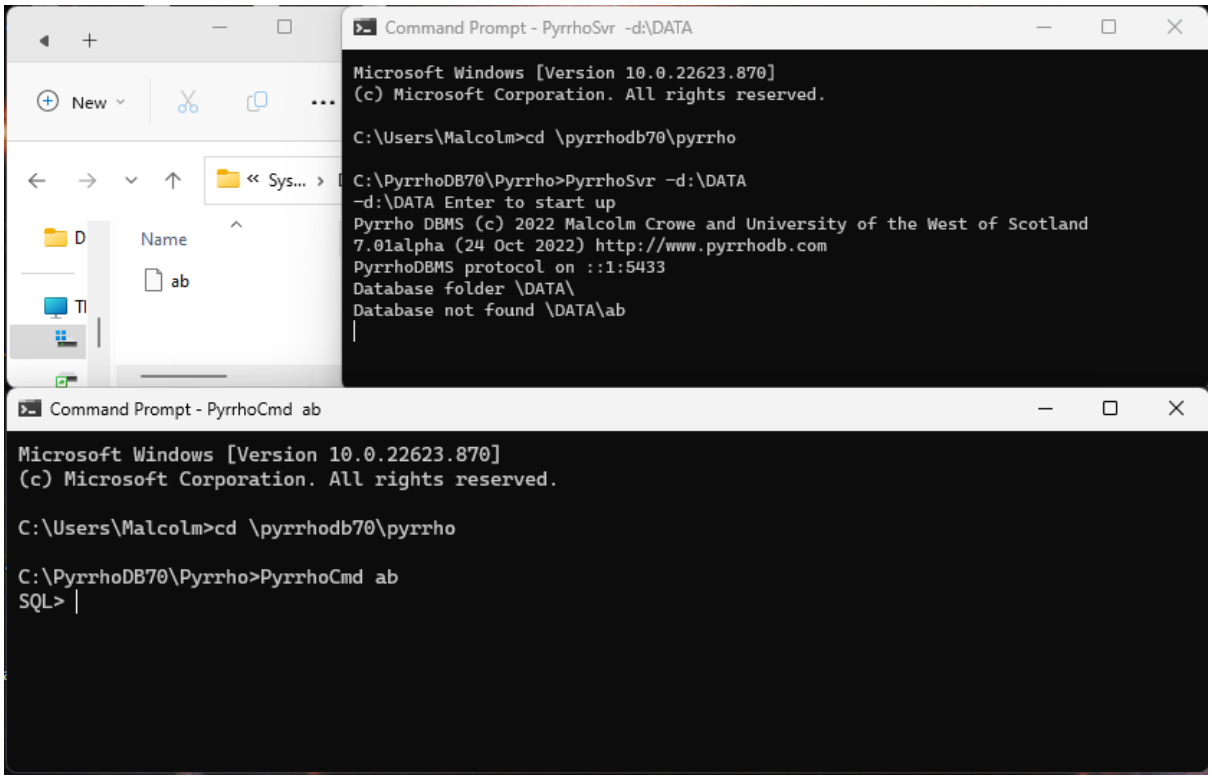
[14 @ 06:58]

The simplest way to get Pyrrho to do work for you is to connect to it using the command line processor PyrrhoCmd, and when you do that you can specify the database you are connecting to. It can be a new database, as here.

`PyrrhoCmd ab`

[15 @ 07:17]

And when you give the command, the server immediately creates an empty database in the database folder.



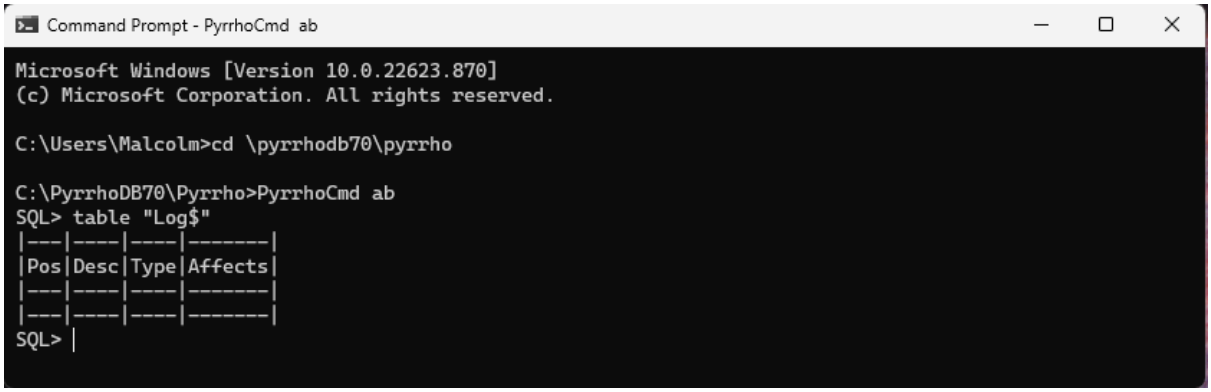
This is a transaction log, and at the moment it contains just 5 bytes, and its opened exclusively by the server, so that we can't look at it unless the server is stopped.

[16 @ 07:38]

We can examine the contents of the transaction log with the table "Log$" statement. Log$ is one of many system tables for inspecting server internals from SQL. It is empty at the moment.

SQL> **`table "Log$"`**

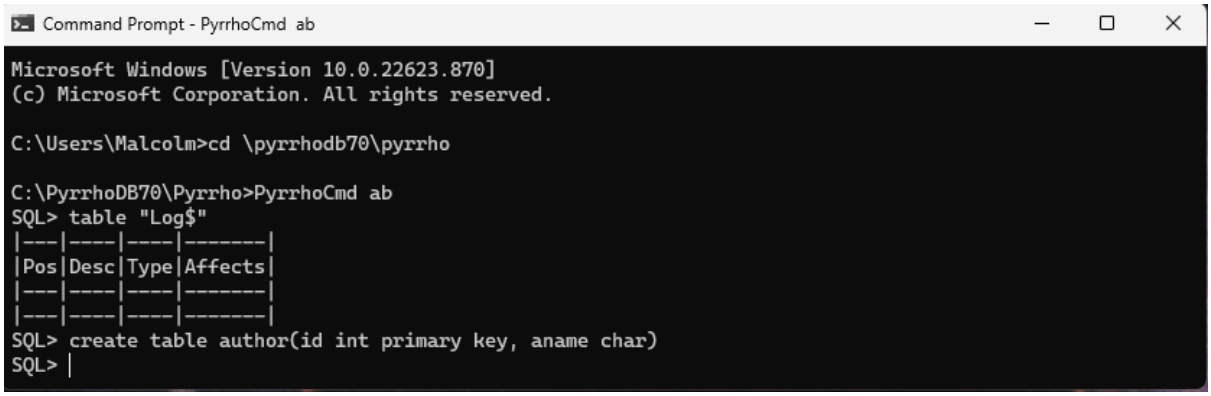
(`table "Log$"` is another way of writing `select * from "Log$"`.) Be careful to use a straight double-quote character here.



[17 @ 07:54]

Let's make a base table in the database ab, by giving a CREATE TABLE command, and this is the normal syntax in the SQL standard, but Pyrrho has it own ideas about primitive types. So CHAR, for example, is an unbounded character string, and INT is actually a "bigint" – it's up to 2000 bits

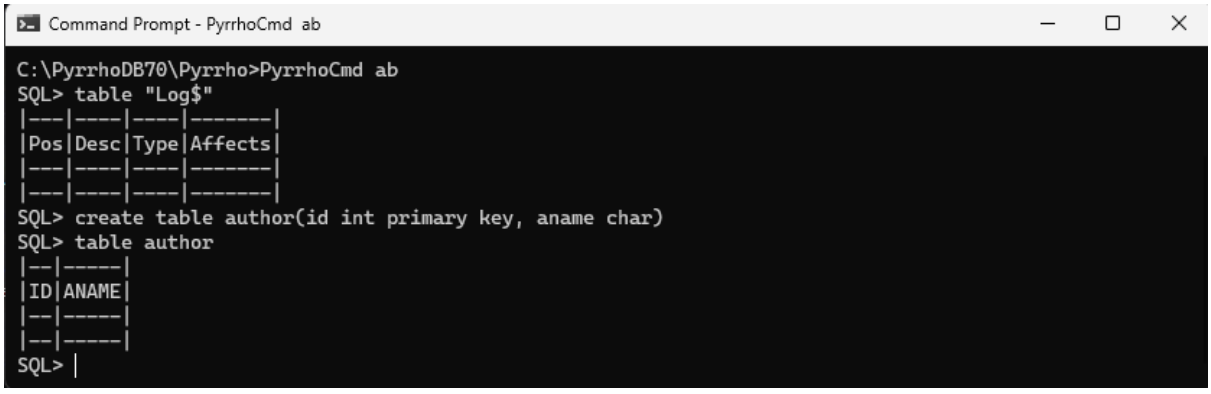
SQL> **create table author(id int primary key, aname char)**



[18 @ 08:20]

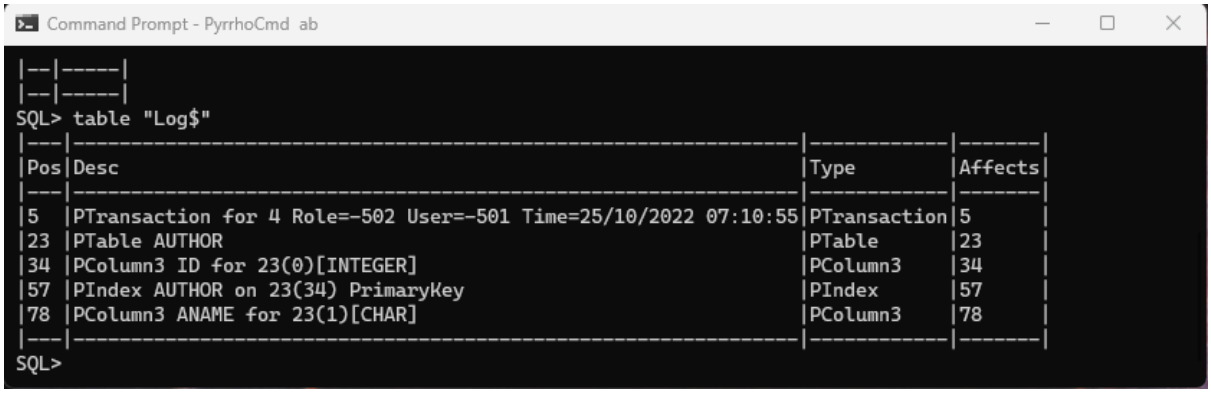
So, we have created the table, but let's look at it: of course it will be empty.

SQL> **table author**



[19 @ 08:29]

Let's examine how this has been represented in the transaction log.



We can see that there are 5 objects in the transaction log, all wrapped in a single transaction. The AUTHOR table is mentioned, and the two column names, and the primary key as defined in the create table request, is defined there along with its key.

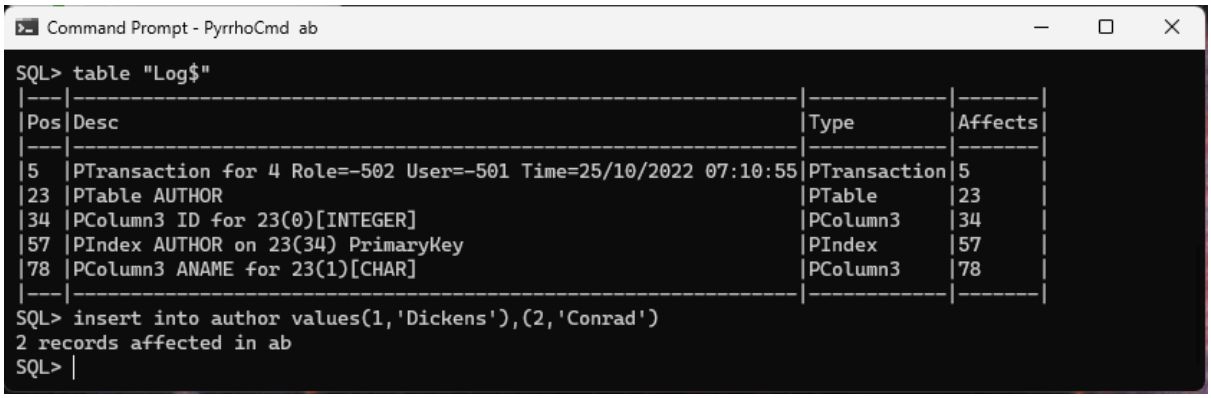
You will notice that the file positions that are here (these are actual byte positions in the file) are used as unique identifiers for objects in the database. Pyrrho uses them throughout: names can be changed but the file position of the definition can't. So here the table AUTHOR is referred to as 23, ID is 34, and that becomes the primary key, and so on. The numbers (0) and (1) give the ordering of the columns in the table.

The first entry for the transaction shows the identity of the user who made the changes, and the role they were using, but in this database, no users have been defined yet, so these are system defaults.

[20 @ 09:38]

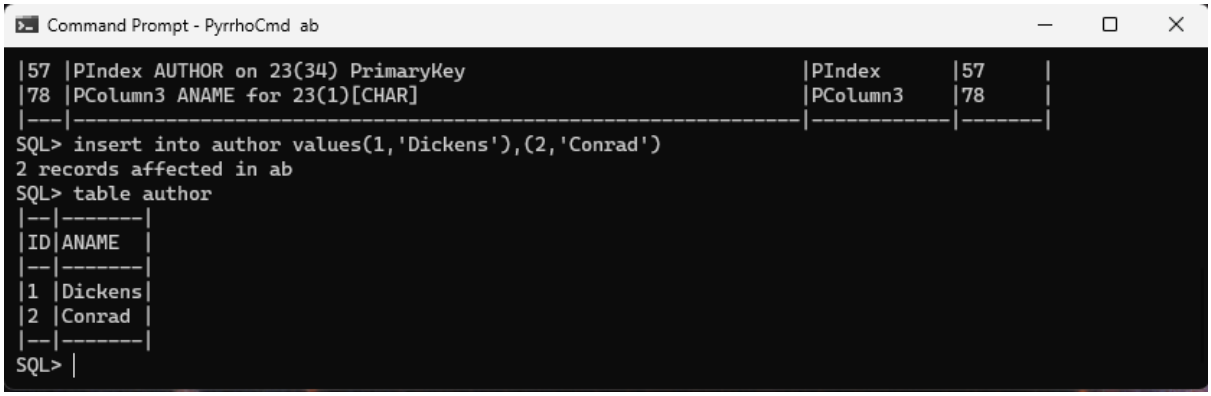
Let us add some rows to the AUTHOR table (be careful to use the straight single-quote character here):

SQL> **insert into author values(1,'Dickens'),(2,'Conrad')**



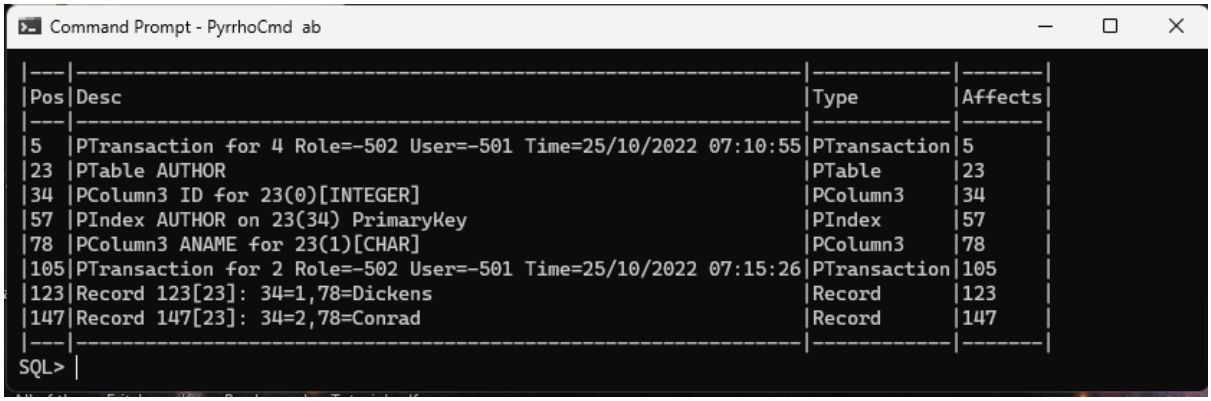
Let us look at our AUTHOR table

SQL> **table author**



[21 @ 09:53]

In the transaction log we see that the auto-committed transaction has both records.



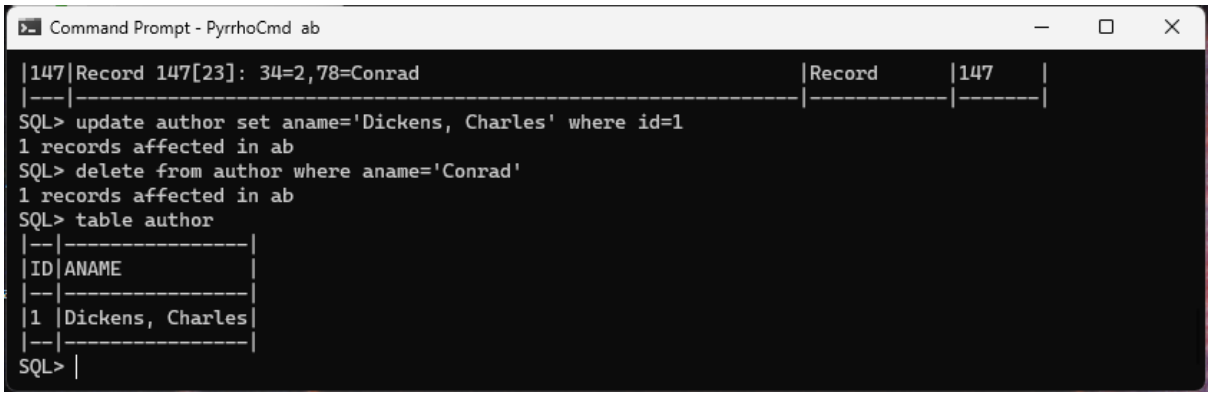
[22 @ 10:04]

We can update, and delete.

SQL> **update author set aname='Dickens, Charles' where id=1**

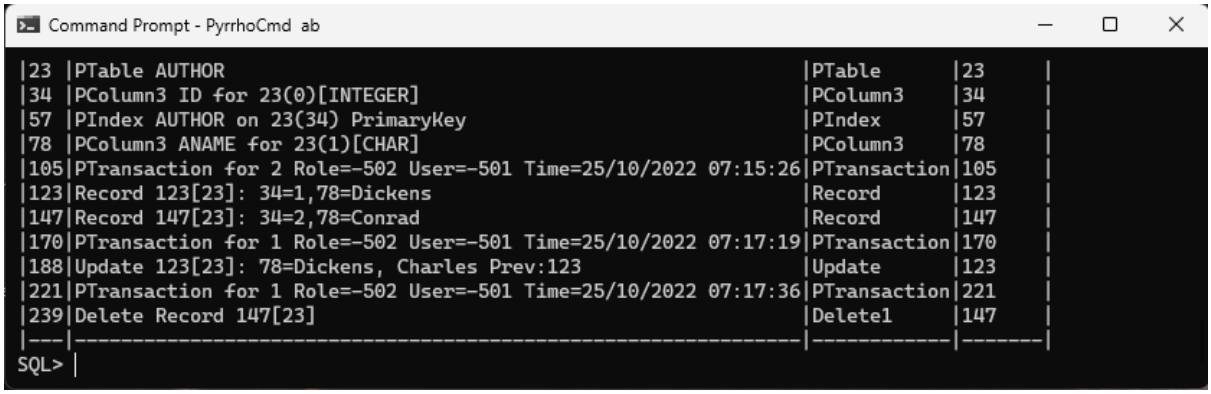
SQL> **delete from author where aname='Conrad'**

SQL> table author



[23 @ 10:18]

Finally let's see the update and delete in the Log.



This completes the demonstration showing the use of the transaction log and the transaction markers.