userman

COLLABORATORS

	TITLE : userman		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 14, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

userman

1.1 userman.guide

```
                SQLdb
*****

This file documents the March, 1993 alpha version of SQLdb.
```

1.2 userman.guide/Copying

```
Copying
*******
```

1.3 userman.guide/Installation

```
Installation
************

Copy the main program sqldb/db to a directory in your executable path.
Add the assignment SQLDB: to your user-startup and have it point to a
central location for your tables.

SQLdb checks your stack on startup to see that it is at least 70000.
This amount of stack is good for two levels of subquery's.  Add 25000
or so for each level of subquery beyond that that you intend to use.
```

1.4 userman.guide/Getting Started

```
                Getting Started
***************

  Command-line options to SQLdb:

  Option  Arguments Default   Meaning
  ----- --------  ------    ------
  -t  'd' or 'r'  'r'   Temporary table storage
  -s  none    NA    Server mode

See
                Temporary table
                . See
                Server Mode
                .
```

1.5 userman.guide/Temporary table

```
Temporary table
===============

Temporary tables are created whenever you give SQL a select query to
perform.

The 'r' argument specifies that temporary tables are to be stored in
main memory.

The 'd' argument specifies that temporary table are to be stored in the
current directory from which SQLdb was started from.

After the query is finished processing, as you might expect, the
temporary tables are removed.
```

1.6 userman.guide/Tutorial

```
                Tutorial
********
```

1.7 userman.guide/Introduction

```
Introduction
============
```

The objective of this tutorial is to introduce you to enough SQL commands and concepts to get you started. For a thorough treatment, I would recommend one of the many books on SQL. One such book is _Using SQL_ by James R. Groff & Paul N. Weinberg, published by McGraw-Hill, ISBN 0-07-881524-X.

The primary function of the SQL language is to support the definition, manipulation, and control of data in a relational database. A relational database is collection of tables. A table is an unordered collection of rows. The terms "file", "record", and "field" in a flat-file database correspond to the relational terms "table", "row", and "column".

In all examples in this tutorial, ‘dbcsh>’ is the command-shell prompt and should not be typed.

1.8 userman.guide/Creating

```
Creating
========
```

The first thing you need to do is create a database. This is accomplished with the ‘CREATE DATABASE’ command:

```
     dbcsh> create database tutorial;
```

Now we are ready to create a table to hold our data. The data will be the venerable address book. We will use the ‘CREATE TABLE’ command illustrated below:

```
     dbcsh> create table address_book (person_id integer,
                                        first_name char(15),
                                        last_name char(40),
                                        address char(50),
                                        city char(20),
                                        state char(2),
                                        zip char(9),
                                        phone char(11)
                                       );
```

As you can see, ‘address_book’ is the name of the table. To specify the
columns of the table you supply a comma separated list of definitions
within parentheses. Each definition consists of the column name and
its data type. So ‘last_name’ is a character column with a size of 40
characters.

For efficient access to specific data, you will want to create an index
on certain columns of your table. Indexes entries must be unique
values. So, if you create an index on ‘first_name’ and ‘last_name’,
you cannot have two people with the same first and last names in this
table.

Most times you need, for example, the phone number of a certain person.
So you would look them up by their name to find the number. So we
will create an index on those columns with the ‘CREATE INDEX’ command:

```
     dbcsh> create index name_idx on address_book (last_name, first_name);
```

A note to non-registered users: You can create indexes and they will be
updated properly. However, they will not be used in query optimization
to speed up your queries.

1.9 userman.guide/Adding and Changing

```
Adding and Changing
===================
```

Now that you have your table, you need to put your data into it. To do
this you use the ‘INSERT INTO’ command:

```
     dbcsh> insert into address_book values (1,'Kyle','Saunders',
                                              '4418 N. 4th. Road',
                                              'Arlington','VA',
                                              '22203','5551212'
                                             );
     dbcsh> insert into address_book values (2,'John','Smith',
                                              '1234 Outer Join Way',
                                              'Relational','NY',
                                              '12345','5557777'
                                             );
```

We should make sure that the rows we just added are really in the
table. The way all rows are retrieved is through the ‘SELECT’
statement:

```
     dbcsh> select * from address_book;
     PERSON_ID  FIRST_NAME  LAST_NAME  ADDRESS             CITY        STATE  ZIP ↩
            PHONE
     ---------  ----------  ---------  ------------------  ----------  -----   ↩
        -----  -------
             1  Kyle        Saunders   4418 N. 4th. Road   Arlington   VA      ↩
           22203  5551212
             2  John        Smith      1234 Outer Join Way Relational  NY      ↩
           12345  5557777
```

Say you forgot where to send the registration fee, so you needed to
look up my address. You would use the ‘SELECT’ command with a ‘WHERE’
clause:

```
     dbcsh> select * from address_book
                     where first_name = 'Kyle'
                       and last_name = 'Saunders';
     PERSON_ID  FIRST_NAME  LAST_NAME  ADDRESS             CITY        STATE  ZIP ↩
            PHONE
     ---------  ----------  ---------  ------------------  ----------  -----   ↩
        -----  -------
             1  Kyle        Saunders   4418 N. 4th. Road   Arlington   VA      ↩
           22203  5551212
```

What if you just realized that ‘John Smith’’s phone number is wrong?
Then you need to update the data in the row. So you would use the
‘UPDATE’ statement:

```
     dbcsh> update address_book set phone='5559876'
                   where first_name = 'John'
                     and last_name = 'Smith';
```

Now you decide that you no longer want to talk to ‘John Smith’, so you
want to remove him from the table. You would use the ‘DELETE FROM’
command:

```
     dbcsh> delete from address_book
                   where first_name = 'John'
                     and last_name = 'Smith';
```

1.10 userman.guide/Finishing Up

Finishing Up
============

To close the database, you simply use the ‘CLOSE DATABASE’ command:

```
     dbcsh> close database;
```

To leave the program, you use the command-shell ‘EXIT’ command:

```
     dbcsh> exit;
```

1.11 userman.guide/Command Shell

```
                Command Shell
*************

The command shell is a line oriented user interface.  Commands are
typed and the results are displayed on the screen.  The commands shell
is similar in operation to the Unix csh(1) program.

Commands may span physical lines.  Commands *MUST* be terminated with a
semi-colon `;'.

Upon startup, the command shell will attempt to execute commands in the
file `.dbcshrc'.  If this file does not exist in the current directory,
the command shell will give an error message saying so, and will
continue.

  The following variables have special meaning to the command shell:

  Name    Default   Meaning
  ---   ------    ------

  EDITOR    'vi'    Editor to be used with EDIT COMMAND

  PROMPT    'dbcsh>'  Command shell prompt string

  HISTORY   25    Number of commands in history buffer


                 History Substitution

                 Alias Substitution

                 Variable Substitution

                 Commands
```

1.12 userman.guide/History Substitution

```
History Substitution
====================

Not implemented.
```

1.13 userman.guide/Alias Substitution

```
Alias Substitution
==================

Not implemented.
```

1.14 userman.guide/Variable Substitution

```
Variable Substitution
=====================

Not implemented.
```

1.15 userman.guide/Commands

```
                Commands
========



                 EDIT COMMAND

                 EXIT

                 HISTORY

                 SET

                 SYSTEM

                 VERSION
```

1.16 userman.guide/EDIT COMMAND

```
EDIT COMMAND
------------

‘EDIT COMMAND’ HISTORY-NUMBER

The edit command command lets you edit the specified command using the
editor specified by the EDITOR variable.
```

1.17 userman.guide/EXIT

```
EXIT
----

`EXIT'

The exit command will exit you from the command shell.  All open tables
will be closed for you when you exit.

If you have an open cursor, you will receive an error message and the
program will not exit.
```

1.18 userman.guide/HISTORY

```
HISTORY
-------

`HISTORY'

The history command displays a list of the history buffer.  Each entry
consists of the history number and the command text.
```

1.19 userman.guide/SET

```
SET
---

`SET' [ VARIABLE-NAME = VARIABLE-VALUE ]

The set command lets you store values in command shell variables that
can be used later.  If the arguments to set are omitted, a listing of
all variables and values is produced.
```

1.20 userman.guide/SYSTEM

```
SYSTEM
------

`SYSTEM' SYSTEM-COMMAND

The system command allow you to execute a system command, such as `dir'
or `ls', without having to leave SQLdb.
```

1.21 userman.guide/VERSION

```
VERSION
-------

‘VERSION’

The version command displays the current version and any other
pertinent information.
```

1.22 userman.guide/Server Mode

```
                Server mode
***********

When started up in Server Mode, SQLdb will open up an ARexx port called
‘SQLserver’ and await commands from the port.



                 ExecSQL

                 ShutdownSQL

                 GetLastCode

                 GetLastErrMsg
```

1.23 userman.guide/ExecSQL

```
ExecSQL
=======

‘’‘ExecSQL’‘’ ‘’SQL-COMMAND-STRING;‘’

‘ExecSQL’ will send the given command to the interpreter to be executed.
If the command is a fetch from a cursor, the result variable will
contain the fetched row.
```

1.24 userman.guide/ShutdownSQL

```
ShutdownSQL
===========

‘’‘ShutdownSQL’‘’
```

ShutdownSQL will tell SQLdb to close the ARexx port and quit. The
command will fail if there are any open tables or cursors.

1.25 userman.guide/GetLastCode

```
GetLastCode
===========

`'`GetLastCode'`'
```

Will put the result code from the last command into the result variable.

1.26 userman.guide/GetLastErrMsg

```
GetLastErrMsg
=============

`'`GetLastErrMsg'`'
```

Will put the error message from the last command into the result
variable.