NAME

bibsql − search an SQL (Structured Query Language) database of BibTeX data

SYNOPSIS

bibsql [**−−author**]
[**−−command** *'command1; command2; . . .'*]
[**−−database** *dbname*]
[**−−help**]
[**−−options** *'. . . server options . . .'*]
[**−−server** *[MySQL | psql | PostgreSQL | SQLite]*]
[**−−user** *dbuser*]
[**−−version**]
< infile or *bibfile1 bibfile2 bibfile3 . . .*
> outfile

DESCRIPTION

bibsql is an interface to SQL (Structured Query Language) client programs for searching an SQL database previously created from the conversion of BIBTEX data by **bibtosql**(1). The manual pages of that program describe the format of the SQL tables that contain the BIBTEX data, and some of that description is paraphrased later in the **DETAILED DESCRIPTION** section.

Once the SQL database has been created with the help of **bibtosql**(1), starting **bibsql** and issuing queries is relatively quick: search responses can be produced in a fraction of a second on modern personal and office computers, even for a database with a million or so entries.

OPTIONS

Command-line options may be abbreviated to a unique leading prefix, and lettercase is *not* significant.

The leading hyphen that distinguishes an option from a filename may be doubled, for compatibility with GNU and POSIX conventions. Thus, **−a**, **−author** and **−−author** are equivalent.

−−author Display an author credit on *stdout*, and then terminate with a success return code.

−−command *'command1; command2; . . .'*
Specify one or more SQL commands to be executed in batch mode, after which, the program terminates with a success return code. Batch mode suppresses status messages and table headers and footers.

−−database *name of BIBTEX database*
Specify the name to be used for the database in the SQL system. The default name is *bibtex*, and to avoid user confusion, it is recommended that the name be changed only for database experiments.

−−help Display a help message on *stdout*, giving a usage description, similar to this section of the manual pages, and then terminate with a success return code.

−−options *'. . . server options . . .'*
Specify additional options to be passed verbatim to the SQL server.

For example, with a nondefault database user name, a MySQL server might require a **−p** option to request that a password prompt be issued.

−−server *[MySQL | psql | PostgreSQL | SQLite]*
Select the database server type for which input is to be prepared (default: *SQLite*).

The name *psql* is an alias for *PostgreSQL*, since the former is the name of the client program on many systems.

The name may be abbreviated to any unique leading prefix, and its lettercase is *not* significant.

Although the input formats of the supported SQL systems are similar, there are important small differences that make it imperative to identify the target system.

−−user *name of database user*

Specify the user name with access rights to the SQL database. The default user name is *reader*, which gives read-only access to the database.

This name is unrelated to the Unix login name.

This option exists primarily for experimentation and database management, and should rarely be needed.

−−version Display the program version number and release date on *stdout* and then terminate with a success return code.

DETAILED DESCRIPTION

The *bibtab* table contains BIBTEX document entries with these column names:

`authorcount, editorcount, pagecount, bibtype, filename, label, author, editor, booktitle, title, crossref, chapter, journal, volume, type, number, institution, organization, publisher, school, address, edition, pages, day, month, monthnumber, year, CODEN, DOI, ISBN, ISBN13, ISSN, LCCN, MRclass, MRnumber, MRreviewer, bibdate, bibsource, bibtimestamp, note, series, URL, abstract, keywords, remark, subject, TOC, ZMnumber, entry.`

The first three *count* columns in the *bibtab* table are integers. All of the remaining columns are variable-length strings whose maximum length is at least 65,536 characters.

The *namtab* table contains author and editor names with these columns:

`name, count.`

The count column records the number of occurrences of the name.

The *strtab* table contains BIBTEX `@String{key = value}` definitions with these columns:

`key, value, entry.`

In the tables, the `entry` column contains the original BIBTEX entry, *exactly* as it was input to **bibtosql**(1).

Except in the `entry` column, consecutive whitespace is reduced to a single space, TEX macros are simplified or removed, and TEX accents and braces are stripped from key values. That convention makes specification of search strings for key values easier. Thus, to search for `Paul Erd{\H{o}}s`, use the pattern `%P%Erdos%`. To find documents with `{\TeX}book` or `\TeX{}book` or `\TeX book` in their titles, use the pattern `%texbook%`. Search patterns are described in the next section.

Indeterminate, unknown, or unset values are recorded as NULL, so as to facilitate their exclusion in later searches.

NULL values are never used in string comparisons or in numeric expressions, or in built-in functions that operate on strings or numbers. Thus, if you take the sum or average of a numerical column, only rows with numbers in them participate.

The expressions *column* `IS NULL` and *column* `IS NOT NULL` test for the presence or absence of a NULL value.

While most of the key names in the *bibtab* table are standard ones in BIBTEX, a few are not. They include

- `CODEN` (Chemical Abstracts *CODE Name* for serial publications).
- `DOI` (*Digital Object Identifier*, a *unique and permanent* name for the master copy of an electronic document). If it does not look like an Internet URL, convert it to one by prefixing it with `http://dx.doi.org/`.
- `ISSN` (*International Standard Serial Number*: eight digits, optionally written as two groups of four digits separated by a hyphen).
- `ISBN` (old-style *International Standard Book Number*: 10 digits, the last of which may be *X* or *x*). Optional hyphens separate it into four digit groups that define the country or language, publisher, book

number, and a final check digit for error detection.

- `ISBN13` (new-style (since 2007) *International Standard Book Number*: 13 digits, the last of which may be *X* or *x*). This contains the old ten-digit ISBN, with a new prefix of `978-`, and a revised check digit. When the supply of unused ISBNs is exhausted, a new prefix `979-` will be assigned, and such numbers will then not be convertible to the old ten-digit form.
- `LCCN` (US *Library of Congress Catalog Number*), used for book identification at many US libraries, and at some libraries in other countries.
- `MRclass` (list of American Mathematical Society *Math Reviews* five-character subject classification codes).
- `MRnumber` (*Math Reviews* database number).
- `MRreviewer` (*Math Reviews* reviewer name).
- `bibdate` (Unix **date**(1) string in the formats `Sat Oct 25 20:15:00 MDT 2008` or `Sat Oct 25 20:15:00 2008`; it records a timestamp of the last significant change to the BibTeX entry data).
- `bibsource` (arbitrary text indicating source(s) of bibliographic data, often as a list of Internet URLs or Z3950 library catalog addresses).
- `bibtimestamp` (alternate representation of `bibdate` value in the form `2008.10.25 20:15:00 MDT`, for which lexicographic sort order is also time order).
- `URL` (Internet *Uniform Resource Locator(s)* for retrieval of an electronic form of the document).
- `remark` (additional commentary about the entry that is not intended to be included in a typeset reference list).
- `subject` (subject classification phrases).
- `TOC` (table of contents of the publication).
- `ZMnumber` (European Mathematical Society *Zeitschrift fuer Mathematik* database number).

SQL SEARCH TUTORIAL

The *Structured Query Language*, SQL, is a system that allows relational database lookups to be carried out according to an imperative English-like grammar that is nevertheless precise enough to allow unambiguous interpretation by computer software.

The acronym SQL is commonly pronounced either as its letters (*ess cue ell*), or like the name *sequel*.

SQL is defined in several national, government, international, and industry standards, including ANSI (X3.168-1989, X3.135-1992, 9579-2-1993, 9075-3-1995, and 9075-4-1996), FIPS (127:1990, 127-2:1993, 193:1995), ISO/IEC (9075:1987, 9075:1989, 9075:1992, 9075:2003, and 13249:2007), and X/Open (CAE 1994). Many 2008-vintage SQL systems claim conformance to most of the 1992 ISO Standard.

Database technology has been well optimized since the relational model was first proposed by E. F. Codd in 1970, and rather complex queries can usually be handled quickly by any SQL database. While industrial-strength commercial database systems capable of scaling to enormous sizes are in worldwide use, there are at least three freely-available SQL systems that are distributed either under open-source licenses, or are in the public domain: **mySQL**, **PostgreSQL**, and **SQLite**.

Of the three, SQLite is to be recommended for the small to medium bibliographic database applications supported by **bibtosql**(1), because it is highly portable across all common desktop operating systems and CPU architectures, its internal database formats are platform independent and stored in a *single* host file, and importantly, it requires no special system privileges to install or to operate.

Although the expansion of the acronym SQL suggests that it describes a language that should be the same for all modern databases, there are, alas, small variations in syntax between different systems. In the following, we assume SQLite, since it is likely to be the one most used with **bibtosql** and **bibsql**. At the time of writing this, its executable program is called **sqlite3**.

It is important to observe that SQL is *not* a programming language: there are no variables, no loops, no conditionals, and no user-defined functions or procedures. When programmability is required, it is conventional to embed calls to an SQL interface library, either in a high-level compiled language like C, C++, C#, or Java, or in a scripting language, such as JavaScript, Perl, PHP, Python, or Ruby, all of which have SQLite module interfaces.

Lettercase in SQL commands and keywords is not significant.

Character strings in SQL are written with surrounding single quotes, like this: `'SQL'`. Some SQL clients also support quotation-mark delimiters, `"SQL"`.

To represent a single quote inside a string, double it: `'O''Neil'` is the name *O'Neil*. In MySQL, use a backslash escape instead: `'O\'Neil'`.

Long strings can be split into separate strings that are joined with the double-bar concatenation operator. For example, these two expressions evaluate to the same string:

```
'Aloisius Baldwin Chadwick, IV'
'Aloisius ' || 'Baldwin ' || 'Chadwick, ' || 'IV'
```

Comments in SQLite take two forms. An Ada-style double hyphen starts a remark that continues to end of line or end of file, whichever comes first. MySQL additionally requires that the double hyphen be followed by at least one space to eliminate an ambiguity in the SQL expression grammar. Otherwise, C-style /* ... */ comments can span one or more lines. Comments cannot be nested, and can appear inside commands anywhere that whitespace can (except inside character strings).

To understand the basics of SQL search commands, it is useful to view the database as a tabular array of values, where the rows are indexed by arbitrary (and unspecified) unique integer numbers, and each column is named by a field name given in `CREATE` and `INSERT` commands. The search task is then to narrow the selection of records to just one or more cells of the table by specifying constraints on the values in the columns.

The format of the output depends on the database system. For SQLite, it can be set by a dotted command described in the help system like this:

```
.mode MODE ?TABLE?  Set output mode where MODE is one of:
                      csv    Comma-separated values
                      column Left-aligned columns.  (See .width)
                      html   HTML <table> code
                      insert SQL insert statements for TABLE
                      line   One value per line
                      list   Values delimited by .separator string
                      tabs   Tab-separated values
                      tcl    TCL list elements

.nullvalue STRING   Print STRING in place of NULL values

.output FILENAME    Send output to FILENAME

.output stdout      Send output to the screen

.separator STRING   Change separator used by output mode and .import

.width NUM NUM ...  Set column widths for "column" mode
```

The default is to separate cell values by vertical bars, with each table row output on a single line (unless the string data contain linebreaks). Examples are given later in this section. However, other output styles chosen by the SQLite `.mode` command make it easy to output the data in formats suitable for input to other databases, spreadsheets, and Web pages.

The simplest query asks for a return of all records, where the asterisk means all data, and a final semicolon is required to terminate the command:

select * from bibtab;

```
1||9|article|acm-turing-awards.bib|Perlis:1967:SAS|
Alan J. Perlis|||The Synthesis of Algorithmic Systems||
j-J-ACM|14||1|||||||1--9||jan|1|1967|JACOAH|
http://doi.acm.org/10.1145/321371.321372|||
0004-5411 OR 00045411|||||Mon Dec 05 19:37:58 1994||
1994.12.05 19:37:58 ???||||||This is the 1966 ACM
Turing Award Lecture, and the first award.||||
@Article{Perlis:1967:SAS,
  author =       "Alan J. Perlis",
  title =        "The Synthesis of Algorithmic Systems",
  journal =      j-J-ACM,
  volume =       "14",
  number =       "1",
  pages =        "1--9",
  month =        jan,
  year =         "1967",
  CODEN =        "JACOAH",
  DOI =          "http://doi.acm.org/10.1145/321371.321372",
  ISSN =         "0004-5411",
  bibdate =      "Mon Dec 05 19:37:58 1994",
  acknowledgement = ack-nhfb,
  remark =       "This is the 1966 ACM Turing Award Lecture, and the
             first award.",
}
. . .
```

Here, we wrapped the long first line for readability, and showed only the first record returned. The order of records depends on the database creation and update history, and is unpredictable without further specifications.

Next, we limit the output to just three specified columns, and further limit the selection with a `WHERE` clause:

select year, author, title from bibtab
where author like '%Perlis%'
and year = '1967';

```
1967|Alan J. Perlis|The Synthesis of Algorithmic Systems
1967|B. A. Galler and A. J. Perlis|A proposal for definitions in ALGOL
```

Because the command is long, we wrote it on separate lines.

The `LIKE` keyword is followed by a string wherein *percent* represents zero or more characters, *underscore* a single character, and, in SQLite, lettercase is *ignored*. Use `NOT LIKE` to negate the comparison. To search for a literal percent or underscore, double them in the search pattern.

To make string comparisons case sensitive in SQLite, set a library option like this:

pragma case_sensitive_like = on;

Set it to `off` to restore the default behavior. SQLite recognizes synonyms `true`, `yes`, and `1` for `on`, and `false`, `no`, and `0` for `off`.

There is also a `GLOB` keyword that uses Unix pathname matching, where *asterisk* matches zero or more characters, *question mark* matches a single character, and lettercase is always *significant*. Unfortunately, there is no standard support in SQLite for regular-expression matching like that provided by many other Unix tools and scripting languages, and some other SQL systems.

We can also use string equality tests, but then the match must be exact, including lettercase:

select year, author, title from bibtab
where author = 'Alan J. Perlis'
order by year;

```
1958|Alan J. Perlis|Announcement
1963|Alan J. Perlis|Computation's development critical to our society
1967|Alan J. Perlis|The Synthesis of Algorithmic Systems
1969|Alan J. Perlis|Introduction to extensible languages
1978|Alan J. Perlis|The American side of the development of Algol
1986|Alan J. Perlis|Two Thousand Words and Two Thousand Ideas --- The 650 at Carnegie
```

In the **order by** clause, the operand can be a column name or an ordinal number: **order by 1, 3** sorts by the first column, and when that column has the same values, by the third column.

Such a search is likely to miss many entries belonging to alternate spellings of the selected author, such as these:

select year, author, title from bibtab
where author = 'A. J. Perlis'
order by year;

```
1964|A. J. Perlis|A format language
1964|A. J. Perlis|Programming of digital computers
1964|A. J. Perlis|How should ACM publish computer research?
1966|A. J. Perlis|A Forum on Algorithms: A new policy for algorithms?
1975|A. J. Perlis|Introduction to Computer Science
1981|A. J. Perlis|The American side of the development of ALGOL
```

It also misses entries where Perlis is one of multiple authors, since SQL string-equality tests always compare against the full string values.

Use the `namtab` table to find the frequencies and variations of an author or editor name in the database:

select count, name from namtab
where name like '%Steele%'
order by 1 desc;

```
15|Guy L. Steele Jr.
3|Guy L. Steele
2|Guy L. Steele, Jr.
1|G. L. Steele, Jr.
1|G. Steele
```

A more complex query requests unique output from a range of years, sorts the data in descending order, and limits the number of records returned by the command to just five:

select distinct year, author, title from bibtab
where author like '%D%Knuth'
and '1955' < year
and year < '1970'
order by year desc
limit 5;

```
1969|Donald E. Knuth|Seminumerical Algorithms
1968|Donald E. Knuth|Very magic squares
1967|Donald E. Knuth|The Remaining Trouble Spots in ALGOL 60
1966|Donald E. Knuth|Errata: ``Additional comments on a problem in ...''
1966|Donald E. Knuth|Letter to the Editor: Additional comments on a ...
```

Some SQL systems permit the quotes around the year values to be omitted, but strictly, they are strings, not integers, since they occasionally contain a range or list of years.

The expression **'1955' < year and year < '1970'** can also be written as **year between '1956' and '1969'**: the endpoints of the **between** operator are included in the range test.

When multiple logical operators are used in an expression without disambiguating parentheses, **and** is evaluated before **or**. Thus, **a and b or c and d** is treated as if it were written **(a and b) or (c and d)**. When in doubt about the meaning of a complex expression, parenthesize!

The **select** command can be used for rudimentary expression evaluation in SQL, simply by omitting cell selections. Numerical expressions are evaluated in floating-point arithmetic if at least one of the operands contains a decimal point:

```
select 'ABC' > 'DEF';
0

select 'ABC' < 'ABCDEF';
1

select 1.0 / 3.0;
0.333333333333333

select 1.0 / 3;
0.333333333333333

select 1 / 3;
0
```

In the last example, 1/3 is treated as an integer division, producing a zero result.

The larger SQL systems offer a wide range of numerical and string functions, similar to those of many programming languages. However, SQLite has only a minimal repertoire of built-in functions. Here are examples of some of the SQL functions that can operate on the returned results, including ones that just report counts, averages, extrema, and sums:

```
select count(*) from bibtab;
417349

select count(length(title)) from bibtab where length(title) > 250;
772

select round(417349 / 772);
540.0

select avg(length(title)) from bibtab where length(title) > 0;
61.3055823817683

select max(length(title)) from bibtab;
1746

select round(avg(pagecount)) from bibtab where pagecount > 0;
46.0

select max(pagecount) from bibtab;
4412

select max(editorcount) from bibtab;
18

select max(authorcount) from bibtab;
```

```
115
```

select max(authorcount) from bibtab where bibtype = 'book';

```
23
```

select min(length(entry)) from bibtab;

```
101
```

select max(length(entry)) from bibtab;

```
19269
```

select avg(length(entry)) from bibtab;

```
770.447505564887
```

select sum(pagecount) from bibtab where pagecount > 0;

```
13415587
```

select distinct count(month), lower(month) from bibtab
where length(month) = 3
group by lower(month)
order by cast(monthnumber as number);

```
19111|jan
15411|feb
20423|mar
19398|apr
17489|may
21585|jun
19537|jul
16577|aug
20039|sep
18896|oct
16859|nov
20924|dec
```

select count(publisher), publisher from bibtab
where length(publisher) > 0
group by publisher
order by count(publisher) desc
limit 10;

```
5679|pub-SV
2154|pub-ORA
1820|pub-PROJECT-GUTENBERG
1593|pub-IEEE
1307|pub-AW
1076|pub-PH
891|pub-WILEY
838|pub-ACM
524|pub-PHPTR
425|pub-MCGRAW-HILL
```

select count(journal), journal from bibtab
where length(journal) > 0
group by journal

```
   order by count(journal) desc
   limit 10;
60788|j-LECT-NOTES-COMP-SCI
14540|j-J-MATH-PHYS
9988|j-CACM
8594|j-APPL-MATH-COMP
7896|j-SIGPLAN
7253|j-LINEAR-ALGEBRA-APPL
6435|j-THEOR-COMP-SCI
5344|j-COMPUTER
5335|j-MATH-COMPUT
4974|j-INFO-PROC-LETT

select count(*) from bibtab where authorcount > 0;
406451

select round(100 * count(authorcount) / 406451), authorcount from bibtab
   where authorcount > 0
   group by authorcount
   order by count(authorcount) desc
   limit 5;
48.0|1
28.0|2
13.0|3
5.0|4
1.0|5
```

The rule that NULL values are excluded from consideration in expressions does not apply to the special case of the `count(*)` expression, because it just counts rows, and no row is completely NULL (recall that rows have hidden row numbers).

Here, we discovered several facts about the bibliographic data in this collection:

- There are 417 349 unique BIBTEX entries.
- The longest title is 1746 characters, and 772 documents (about 1 in 540) have titles longer than 250 characters.
- The average nonempty title is about 61 characters long.
- The average document page count is about 46 pages.
- The longest document has 4412 pages.
- There are no more than 18 editors or 115 authors of a single document.
- No book entry has more than 23 authors.
- Entry lengths average about 770 characters, but range from about 100 to almost 20 000 characters.
- The documents described by the BIBTEX entries in the database represent a corpus of more than 13.4 million pages.
- The last four searches are complex queries that use the SQL `GROUP BY` feature to simultaneously compute counts of publishers and journals, producing a list of the top ten that occur most frequently in the database, and do a similar computation to find the percentage of publications with one, two, . . . , five authors.

To avoid the need to first compute the count of documents with authors, the last pair of queries can be rephrased as a single command with an embedded `SELECT` command, but we now make a further restriction of the documents to count only journal articles:

```
select round(100 * count(authorcount) /
        (select count(*) from bibtab
            where authorcount > 0 and
                bibtype = 'article')),
        authorcount
  from bibtab
  where authorcount > 0 and
      bibtype = 'article'
  group by authorcount
  order by count(authorcount) desc
  limit 5;
47.0|1
29.0|2
14.0|3
5.0|4
1.0|5
```

BIBTEX string abbreviations are commonly used for data that are repeated in many entries, particularly for journals, institutions, organizations, publishers, and addresses. The abbreviations can be retrieved from the *strtab* table with searches like these:

```
select entry from strtab where key like 'pub-WILEY%';
@String{pub-WILEY                = "Wiley"}
@String{pub-WILEY:adr            = "New York, NY, USA"}
@String{pub-WILEY-INTERSCIENCE  = "Wiley-In{\-}ter{\-}sci{\-}ence"}
@String{pub-WILEY-INTERSCIENCE:adr = "New York, NY, USA"}

select entry from strtab where value like '%Boston%' order by entry;
@String{pub-ALLYN-BACON:adr     = "Boston, MA, USA"}
@String{pub-AP-PROFESSIONAL:adr = "Boston, MA, USA"}
...
@String{pub-LITTLE-BROWN:adr    = "Boston, Toronto, London"}
@String{pub-MORGAN-KAUFMANN:adrbo = "Boston, MA, USA"}

select count(value) from strtab where value like '%New York%';
59

select distinct key from strtab where key like '%BIRK%' order by key;
pub-BIRKHAUSER
pub-BIRKHAUSER-BOSTON
pub-BIRKHAUSER-BOSTON:adr
pub-BIRKHAUSER:adr
```

It is possible, although complex, to combine searches in multiple tables to collect output data from all of them. Here is just one example, which also introduces another SQL feature of providing short aliases for database and table names within a single query:

```
.separator "\n"

select s.entry, t.entry, b.entry
    from bibtab b, strtab s, strtab t
    where
        b.author like '%Robbins%'
    and b.author like '%Beebe%'
```

and b.title like '%classic%'
and b.publisher = s.key
and b.address = t.key;

```
@String{pub-ORA-MEDIA           = "O'Reilly Media, Inc."}
@String{pub-ORA-MEDIA:adr       = "1005 Gravenstein Highway North,
                                   Sebastopol, CA 95472, USA"}

@Book{Robbins:2005:CSS,
  author =       "Arnold Robbins and Nelson H. F. Beebe",
  title =        "Classic Shell Scripting",
  publisher =    pub-ORA-MEDIA,
  address =      pub-ORA-MEDIA:adr,
  pages =        "xxii + 534",
  year =         "2005",
  ISBN =         "0-596-00595-4",
  ISBN-13 =      "978-0-596-00595-5",
  LCCN =         "QA76.76.O63 R633 2005",
  bibdate =      "Tue Jul 12 16:13:16 2005",
  URL =          "http://www.oreilly.com/catalog/shellsrptg/",
  acknowledgement = ack-nhfb,
}
```

Changing the output separator to a newline suppressed unwanted additional output, producing output that can be copied directly into a BIBTEX file.

Turn command-time reporting on or off like this:

.timer on
.timer off

To list the column fields in the SQLite database, and to get further help, use the commands

.schema bibtab
.help

The output of `.schema` is similar to the `CREATE TABLE` command shown in the manual pages for **bibtosql**(1).

For more on the command syntax of SQLite, consult its Web site documentation collection:

`http://www.sqlite.org/docs.html`

There are hundreds of books on SQL, and many of them are recorded in a BIBTEX bibliography available at this Web location:

`http://www.math.utah.edu/pub/tex/bib/sqlbooks.html`

MySQL NOTES

Unlike SQLite, MySQL is a client/server database system. The biggest part of the software, and all of the data, is on the server side, which can be running on the same machine as the client, or anywhere else as long as there is a network connection between the two programs, and the server allows connections from the client's computer. The client part is what the user interacts with, and if desired, it can be hidden behind a window interface or in a Web browser script. We do not do so with **bibsql**, because graphical interfaces generally destroy the great power of user programmability.

The server attaches ownership and access permissions to various parts of the data, and as a result, setup and management are much more complex compared to a simple system like SQLite. Servers can also make use of an advanced SQL feature called *triggers*. These are software routines that are invoked when data are accessed or modified. They can be used for things like data validation, accounting, and alerting management to unauthorized attempts to access data.

After the **bibsql** script is installed, it may be necessary to make minor edits to it to provide the information needed to connect the clients to the servers: that information is not standardized, and is strongly site dependent. Users of the script are completely isolated from these issues.

In MySQL, `key()` is a built-in function whose name conflicts with a column name in the `strtab` table. You can still use the name, but it must be enclosed in back quotes:

select ‘key‘ from strtab where ‘key‘ like ’%WILEY%’;

Apart from this minor nuisance, all of the SQLite command examples given earlier also work in MySQL, except that the SQLite `GLOB` command is not supported in MySQL.

Conventional Unix-style regular-expression matching is available in MySQL with the `REGEXP` operator, like this:

select year, author from bibtab where author regexp ’Sh[ae]r[ie]+’;

```
+------+--------------+
| 2008 | Shari Trewin |
+------+--------------+
```

The `REGEXP` operator can also be written `RLIKE`.

Unlike normal SQL pattern matching, MySQL regular-expression patterns can match any substring of a value, so there is no need to surround the pattern with `.*` to force a match against the entire string value.

To list the column fields in the MySQL database, and to get further help, use the commands

describe bibtab;
help
help *command*

MySQL normally prints a timing report after the output of each command. You can get detailed performance reports for queries with the MySQL profiling feature. Turn on profiling, run one or more commands, list their query numbers, and then request a report on a particular one:

set profiling = 1;
. . . arbitrary commands here . . .
show profiles;
show profile for query 2;

Besides the many books listed in the `sqlbooks` bibliography cited earlier, extensive documentation about MySQL can be found at its Web site:

```
http://dev.mysql.com/doc/#manual
http://downloads.mysql.com/docs/
http://dev.mysql.com/doc/refman/5.1/en/faqs.html
```

PostgreSQL NOTES

Like MySQL, PostgreSQL is also a client/server system, and has similar problems of considerable management complexity. The human software installer needs to customize the **bibsql** shell script to meet local conventions for connecting SQL clients to their servers. However, users of the script need not be concerned with those issues.

All of the SQLite command examples also work in PostgreSQL, except that the SQLite `GLOB` command is not supported in PostgreSQL, and the `LIKE` operator does *not* ignore lettercase.

Conventional Unix-style regular-expression matching is available in PostgreSQL with the tilde operator, like this:

select year, author from bibtab where author ˜ ’Sh[ae]r[ie]+’;

```
 year |    author
------+--------------
 2008 | Shari Trewin
(1 row)
```

Matching with the tilde operator is lettercase sensitive. Use the companion tilde-star, `~*`, operator for matching with lettercase ignored.

Unlike normal SQL pattern matching, PostgreSQL regular-expression patterns can match any substring of a value, so there is no need to surround the pattern with `.*` to force a match against the entire string value.

The output format can be changed with the help of two PostgreSQL commands:

\pset format *unaligned* | *aligned* | *html* | *latex* | *troff-ms*

The default mode is *aligned*, as in the preceding search example.

The *unaligned* option produces data separated by the current field separator (set by the `\f` command, with a default of a vertical bar).

The other formats produce data suitable for embedding in other documents in the indicated languages. The output data in those formats have insufficient surrounding markup to be used as standalone documents.

\pset border *0* | *1* | *2*

Border style 0 replaces the field separator with whitespace between cell values. Border style 1 is the default, and uses the field separator. Border style 2 produces additional boxing around the output table.

Get further help with these commands:

\?
\h
\h *command*

List the column fields in a PostgreSQL table:

\d strtab

```
 key    | text |
 value  | text |
 entry  | text | not null
```

List available databases:

\l

```
        List of databases
   Name    |  Owner   | Encoding
-----------+----------+----------
 bibtex    | bibtex   | UTF8
 postgres  | postgres | UTF8
 template0 | postgres | UTF8
 template1 | postgres | UTF8
 test      | postgres | UTF8
```

List database users and their permissions:

\du

```
                               List of roles
 Role name | Superuser | Create role | Create DB | Connections | Member of
-----------+-----------+-------------+-----------+-------------+-----------
 anonymous | no        | no          | no        | no limit    | {}
 bibtex    | no        | no          | yes       | no limit    | {}
 postgres  | yes       | yes         | yes       | no limit    | {}
```

Find the type and number of arguments accepted by a build-in function:

\da avg

```
                        List of aggregate functions
   Schema   | Name | Result data type | Argument data types | Description
------------+------+------------------+---------------------+-------------
 pg_catalog | avg  | double precision | real                |
 pg_catalog | avg  | double precision | double precision    |
 pg_catalog | avg  | interval         | interval            |
 pg_catalog | avg  | numeric          | bigint              |
 pg_catalog | avg  | numeric          | numeric             |
 pg_catalog | avg  | numeric          | integer             |
```

```
pg_catalog | avg   | numeric           | smallint           |
```

Toggle command timing on or off:

\timing

Besides the many books listed in the `sqlbooks` bibliography cited earlier, extensive documentation about PostgreSQL can be found at its Web site:

```
http://www.postgresql.org/docs/manuals/
http://www.postgresql.org/files/documentation/pdf/
http://www.postgresql.org/docs/faqs.FAQ.html
```

CAVEATS

SQL has dozens of commands and many powerful expression features that we have not mentioned here. **bibsql** completes its job by starting the requested, or default, SQL client program, and does *no* input filtering whatsoever. The full power of SQL is therefore available to the knowledgeable user. Because some SQL commands can alter or destroy data, or add new data, the shared database itself should be made *read-only*. It is recommended that database updates be allowed only for one or more local database administrators.

At sites with active BIBTEX files, it may be helpful to install an automated job that updates the master database at suitable intervals from lists of BIBTEX directories or files.

FILES

bibsql has no files of its own. The SQL database system manages the bibliographic data, which is *read-only* for **bibsql** users. However, each of the supported database systems normally records a command history file in the user's home directory that may be a useful record of past SQL activity:

`$HOME/.mysql_history`	MySQL command history file
`$HOME/.psql_history`	PostgreSQL command history file
`$HOME/.sqlite_history`	SQLite command history file

The history file is loaded when the SQL client starts, so commands from previous sessions are readily available. Use the up and down arrow keys, or control keys C-p and C-n, to navigate in the history list. Use left and right arrow keys, or control keys C-f and C-b, to move within a single line. The DELete and Backspace keys delete backward, and the control key C-d deletes forward. C-u erase to beginning of line, and C-k erases to end of line, both from the current position. Typing ordinary characters inserts them in the line, and a RETurn key executes the command on the current line.

AUTHOR

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
Tel: +1 801 581 5254
FAX: +1 801 581 4148
Email: `beebe@math.utah.edu`, `beebe@acm.org`, `beebe@computer.org` (Internet)
URL: `http://www.math.utah.edu/~beebe`

The master archive for the **bibsql** and **bibtosql**(1) software is at these equivalent locations:

```
ftp://ftp.math.utah.edu/pub/tex/bibsql
http://www.math.utah.edu/pub/tex/bibsql
```

SEE ALSO

bib2xml(1), **bib2xtx**(1), **bibcheck**(1), **bibclean**(1), **bibdestringify**(1), **bibdup**(1), **bibextract**(1), **bibindex**(1), **bibjoin**(1), **biblabel**(1), **biblex**(1), **biblook**(1), **biborder**(1), **bibparse**(1), **bibsearch**(1), **bibsort**(1), **bibsplit**(1), **bibtex**(1), **bibtosql**(1), **bibunlex**(1), **cattobib**(1), **citefind**(1), **citesub**(1), **citetags**(1), **date**(1), **latex**(1), **mysql**(1), **psql**(1), **ref2bib**(1), **scribe**(1), **sqlite3**(1), **tex**(1), **xml2bib**(1),

xml2wordbib(1), **xtx2bib**(1).