Benchmarking bibsql

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
Email: beebe@math.utah.edu, beebe@acm.org,
beebe@computer.org
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148

17 March 2011

Abstract

This report summarizes the results of benchmarking bibsql with SQL servers running on SAS and SCSI disk storage, and on Fusion-io solid-state storage.

1 Introduction

The bibsql[1] utility [1] provides a Structured Query Language (SQL) interface to data stored in, and served by, three commonly-used, and freely-available, relational databases: MySQL [6], PostgreSQL [5], and SQLite [2].

At this author's site, bibsql provides access to the TEX User Group (TUG) collection of more than a half-million bibliographic records in BIBTEX markup covering major areas of computer science, computer hardware and software, cryptography, history and philosophy of science, markup languages, networks, numerical mathematics, SQL, symbolic algebra, typesetting, Unix, and selected others. The data have largely been prepared, and continue to be maintained and updated, by this author, and are freely available on the Web.[2]

Of the three supported databases, MySQL usually has the highest performance, and SQLite the lowest.

[1] Article and software are freely available at http://www.math.utah.edu/pub/bibsql.

[2] See http://www.math.utah.edu/pub/tex/bib/index-table.html.

Both MySQL and PostgreSQL are client/server databases that require considerable effort to administer, create, manage, and secure. They also need constantly-running server daemons on a machine with adequate CPU, memory, and I/O resources.

By contrast, SQLite requires only a *single* database file that is byte-order and architecture independent, and is easily created from a stream of BIBTEX files by the `bibtosql` utility that is part of the `bibsql` distribution. SQLite needs *no* database administration, *no* running daemon, and *no* network. Its only security issue is handled trivially by normal filesystem protections: if the database file is made read-only, and owned by a user other than that of the `bibsql` user, then it cannot be modified. Read-only status can also be achieved by storage of the database file on write-once media, such as CD-ROM or DVD.

2 A bit about BIBTEX

The BIBTEX system is part of all TEX distributions, and like them, is freely available. BIBTEX and TEX have been ported to all major computing platforms, including desktops, micros, minis, mainframes, and supercomputers. At the TUG'2010 conference in San Francisco, an AT&T Bell Laboratories researcher reported a successful port of the TEX system to the Apple iPhone (cell phone) and iPad (tablet).

BIBTEX files are plain text files, and are understandable even to English-literate humans who have never seen a BIBTEX entry before. A preamble for communicating with TEX, some string definitions, and two sample entries give a flavor of the markup:

```
@Preamble{
    "\input texnames.sty"
  # "\def \TUB {TUGboat}"
}

@String{j-TUGboat                = "{\TUB{}}"}
@String{pub-AW                   = "Ad{\-d}i{\-s}on-Wes{\-l}ey"}
@String{pub-AW:adr               = "Reading, MA, USA"}

@Article{Beebe:2009:BMR,
  author =       "Nelson H. F. Beebe",
  title =        "{{\BibTeX}} meets relational databases:
                 Dedicated to the memory of {Edgar Frank
                 ‘‘Ted’’ Codd (1923--2003)} and {James
                 Nicholas ‘‘Jim’’ Gray (1944--2007)}",
  journal =      j-TUGboat,
  volume =       "30",
  number =       "1",
  pages =        "252--271",
  year =          "2009",
  bibdate =       "Fri Oct 23 16:59:25 2009",
  note =          "TUG 2009 Conference Proceedings volume.",
  acknowledgement = ack-nhfb,
  pagecount =     "20",
  remark =        "Submitted 22 November 2008.",
}

@Book{Knuth:2011:ACP,
  author =        "Donald Ervin Knuth",
  title =         "The Art of Computer Programming: Volume 4,
                  {Combinatorial} algorithms. {Part 1}",
  volume =        "4A",
  publisher =     pub-AW,
  address =       pub-AW:adr,
  pages =         "xv + 883",
  year =          "2011",
  ISBN =          "0-201-03804-8",
  ISBN-13 =       "978-0-201-03804-0",
  LCCN =          "QA76.6 .K64 2011",
  bibdate =       "Fri Mar 4 17:53:38 MST 2011",
  bibsource =     "z3950.gbv.de:20011/gvk",
  series =        "The art of computer programming",
  acknowledgement = ack-nhfb,
}
```

Each document entry begins with `@`, followed by a document type, then a left brace, a label used to cite the entry, a comma, a series of field/value assignments in any convenient order, followed by a right brace. Field values are normally quoted or braced strings, but may also be string-abbreviation names that are frequently used to standardize certain fields, such as journal names, publishers, and addresses. Braces inside value strings protect against the letter downcasing of many bibliographic styles. Whitespace in BIBTEX entries adds readability for humans, but, outside of field values, is insignificant.

When the document is typeset, each TEX (or LATEX) run produces a new auxiliary file that contains information about the document, including sectional titles, citation keys, bibliography style, and bibliography database names. A subsequent BIBTEX run reads the auxiliary file, finds the cited entries in one or more database files, formats them according to the specified bibliography style, and outputs a reference list in TEX markup. That file is read on the next typesetting run to produce the reference list that normally appears at the end of the typeset document.

Although about a dozen field names are standardized, and recognized by all BIBTEX styles, BIBTEX does not complain if a field name is not defined in the selected style file. Some recent style files have been extended to recognize several new field names that have proved useful for recording values for reference lists, or for bibliographic searching. The new fields include abstract, CODEN (Chemical Abstracts periodical number), ISSN (International Standard Serial Number), ISBN (International Standard Book Number — original 10-digit style), ISBN-13 (new 13-digit style), keywords, LCCN (US Library of Congress call number), remark, and subject. Indeed, more than 500 different field names have been used in the TUG bibliography collection.

One particular new field is important for SQL database use: bibdate. It records the date and time of the last important update to the entry, in the form produced by the Unix (and POSIX) `date` command. `bibtosql` parses its value, and converts it to a timestamp of the form `2011.03.04 17:53:38 MST` that is convenient for specifying SQL search ranges. The timezone is optional. For example, an SQL query could include the phrase `bibtimestamp > '2010.01.01 00:00:00 AAA'` to select entries created, or last modified, in the year 2010 or later.

3 Benchmark hardware and software

The server on which the client benchmarks were run, and the Fusion-io[3] storage was installed, is a 2007-vintage Sun Fire X4600 M2 x64 with four AMD Opteron 8218 2.6GHz dual-core CPUs, 16GB DDR2-667 RAM, and two 73 GB SAS drives.

The remote MySQL server is a 2005-vintage Sun Fire V40z with four AMD Opteron 850 single-core 2.4GHz CPUs, 8GB DDR1/333 RAM, and two 74GB 10K rpm SCSI disks.

The remote PostgreSQL server is a 2006-vintage Dell PowerEdge 3250 with two Intel Itanium-2 1.4GHz CPUs, 4GB RAM, and two Maxtor Atlas 10K4_36SCA 60GB SCSI disks.

The operating system on all three servers is Red Hat Enterprise Linux Client release 5.5 (Tikanga).

The SQL client software versions are:

```
% bibsql --version
This is bibsql version 0.02 of 26-Oct-2010

% mysql --version
mysql Ver 14.12 Distrib 5.0.67, for redhat-linux-gnu (x86_64)
using EditLine wrapper

% psql --version
psql (PostgreSQL) 9.0.0
contains support for command-line editing

% sqlite3 --version
3.7.2
```

[3]See `www.fusionio.com`.

Because `bibsql` is a simple wrapper around the various database clients, it has no effect on their performance. All input goes directly to the client program, and the sole function of the wrapper is to simplify the interface by supplying username, database name, and SQL connection information.

4 SQL benchmark commands

One of the important maintenance tasks for large collections of bibliographic data is ensuring correctness and consistency of the data. Correctness guarantees are difficult: while publisher Web sites are now a common source of data that are converted automatically by various software tools to BibTeX form, the original data were entered by humans, and errors of typing or omission are likely to be present. Bibliographic tools are available to allow merging of BibTeX data from multiple independent sources, and reporting discrepancies in field values. For journal bibliographies, additional checks for missing volumes or issues, overlapping page ranges, or unusual page gaps, help to identify errors.

With the power of SQL searches, further checks are possible on BibTeX entries after they have been entered into an SQL database, and a portion of those checks have been used for the queries made for the benchmarks documented in this report.

About two dozen separate tests are present in the `sanity-checks.sql` file that is run several times a year to detect and report unusual data. We show only a few of them here:

```
-- Find files and entries that are missing ISBN-13 data
select filename, label from bibtab
       where isbn is not null
         and isbn13 is null
       order by filename, year, label;

-- Find files and entries that have confused ISBN-13 data
-- (e.g., ISBN-10 value in ISBN-13 field)
select distinct filename from bibtab
        where isbn13 regexp '^[0-9Xx]{10}$'
           or isbn13 regexp '^[0-9Xx]{10}[^0-9]'
           or isbn13 regexp '[^0-9Xx][0-9Xx]{10}$'
           or isbn13 regexp '[^0-9Xx][0-9Xx]{10}[^0-9Xx]'
        order by filename;

-- Find files and entries with old-style 5-character CODENs
select filename, label from bibtab
       where coden regexp '^.....$'
       order by filename, year, label;
```

```
-- Find files and entries with entries containing missing years
select filename, label from bibtab
       where year is null
       order by filename, year, label;

-- Find files and entries with entries containing suspect years
select filename,label from bibtab where
           ( ( substring(year, 1, 4) != '19xx' ) and
             ( substring(year, 1, 4) != '20xx' ) )
       and ( ( substring(year, 1, 4) < '1492' ) or
             ( '2012' < substring(year, 1, 4) ) )
       order by filename, year, label;

-- Find missing bibdate in recent entries
select filename,label from bibtab
       where year >= '2010'
         and bibdate is NULL
       order by filename, label;
```

Because the *S* in SQL means *Structured*, rather than *Standard*, slightly different versions of the sanity checks are needed for each of the backend databases. The queries shown here are for MySQL, and their output is intentionally kept short so that it is insignificant for the benchmark times.

To speed searches, all three databases have separate indexes for each important BIBTEX field. For example, the last sample query requires finding first those entries with years 2010 to date from the year index, and then locating the corresponding null (empty) entries in the bibdate index, and finally reporting their filename and label values, sorted in ascending order. Those two values suffice to identify the defective BIBTEX entries.

Once the defects are repaired in their corresponding BIBTEX files, `cron` jobs that run several times daily for each of the backend databases discover the changed files by virtue of their last-write dates being newer than a timestamp file saved at the end of the most-recent database update. Those jobs invoke `bibtosql` to convert all of the entries in each changed file from BIBTEX format to SQL commands that delete all database entries from those files, then reinsert them. Deletion before insertion is necessary for some of the databases to avoid unnecessary duplication of bibliographic records.

5 Sequential benchmarks

The benchmark of sequential database access was constructed by concatenating 100 copies of the sanity-check file for each database into a single benchmark file. For each backend database, the benchmark first used the disk-based remote SQL server, and then repeated the same queries on the local Fusion-io 280GB solid-state storage device. The only relevant time for database queries

Table 1: Sequential SQL benchmark wall-clock times (minutes) and speedups of solid-state storage over disk.

SQL server	disk	Fusion-io	speedup
MySQL	88.28	75.25	1.17
PostgreSQL	213.88	80.84	2.65
SQLite	2682.88	44.28	60.58

is wall-clock time, because that is the delay suffered by humans. The results are collected in Table 1.

Because only one query from one client is in effect during the benchmark, the SQL server spends most of its time waiting for data from the filesystem. Evidently, SQLite makes inefficient use of the disk, with many small random reads in its 2.13GB database file. The MySQL database tree contains 1.1GB, and the PostgreSQL database tree contains 3.5GB.

For SQLite, the solid-state device is a huge win because it does not have the rotational latency of a magnetic disk. The other two databases have reasonably well-tuned I/O and internal table relations that place fewer demands on the filesystem. Solid-state storage costs several times as much as magnetic disk storage, so for MySQL and PostgreSQL, disk is more cost effective.

The long search times for SQLite show the importance of database software design for achieving high performance. Most uses of that system are for small applications, like managing Web-browser user preferences and caches, where the performance is adequate.

As described in [1], during development of `bibsql`, commercial database backends from IBM (DB2), Ingres, and Microsoft (SQL Express) were evaluated, and rejected because they place severe restrictions on the sizes of database cells, or the number of rows or columns in the database. Some have an alternate cell type that permits unlimited string length, but cells of that type cannot be searched with normal SQL commands. This author had no access to an Oracle database [3] to make a similar evaluation. In a time of low-cost terabyte disk storage and gigabyte computer memories, database products with draconian limits like 8-kilobyte cells deserve no place in the market.

6 Parallel benchmarks

A more realistic benchmark of a multiuser database requires (nearly) simultaneous queries from many separate processes, such as would be experienced by an online order system, or Web-based bibliographic search systems such as those provided by several journal publishers.

We use the same sanity-check data as before, without the 100-fold replication, but this time, we use a shell-script loop to start `MAXTEST` simultaneous background processes:

```
secin=‘date +%s‘
echo "SECONDS IN = $secin"

for n in ‘seq 1 $MAXTEST‘
do
    /usr/local/bin/time bibsql-fusionio -s m \
        < sanity-checks.sql \
        > sanity-checks.out.mysql-fusionio.$n &
done

wait
secout=‘date +%s‘
echo "SECONDS OUT = $secout"
echo "SECONDS ELAPSED = ‘expr $secout - $secin‘"
```

The `wait` command causes the enclosing script to wait until all of the background processes have completed, and then the difference in wall clock times is reported as the elapsed time.

The tests were repeated for various values of `MAXTEST`, and that soon exposed a serious problem with MySQL: its default server configuration allows only 100 simultaneous connections, and each query apparently represents multiple connections, because with 16 parallel `bibsql` jobs, the queries immediately failed with

```
ERROR 1040 (00000): Too many connections.
```

A Web search turned up an explanation, and a simple fix. Edit the local MySQL configurtion file, `/etc/my.cnf`, on each server, and in the `[mysqld]` section, add the assignment

```
max_connections=1000
```

Then, as the `root` user on each server host, restart the MySQL server with

```
# /etc/init.d/mysql restart
```

The restarted MySQL daemons then incorporate the changed limit, a fact that can be verified like this:

```
# mysqladmin variables | grep max_connection
| max_connections   | 1000 |
```

The benchmarks were then resumed and completed successfully. As the number of parallel jobs increased, the PostgreSQL server also reported exceeding a default connection limit. That value was increased from 100 to 1000 in the files `/usr/local/pgsql/data/bibtex/postgresql.conf` and `/fusionio/usr/local/pgsql/data/bibtex/postgresql.conf`, and the servers restarted.

Table 2 summarizes the results for several benchmark runs with varying numbers of parallel queries.

Table 2: Parallel SQL benchmark wall-clock times (seconds) and speedups of solid-state storage over disk.

SQL server	**disk**	**Fusion-io**	**speedup**
	MAXTEST = 8		
MySQL	178	184	0.96
PostgreSQL	240	45	5.33
SQLite	1704	159	10.72
	MAXTEST = 16		
MySQL	376	397	0.95
PostgreSQL	757	135	5.61
SQLite	2261	303	7.46
	MAXTEST = 32		
MySQL	743	775	0.96
PostgreSQL	1052	309	3.40
SQLite	2663	643	4.14
	MAXTEST = 64		
MySQL	1387	1484	0.93
PostgreSQL	2131	583	3.66
SQLite	3542	887	3.99
	MAXTEST = 128		
MySQL	2724	2057	1.32
PostgreSQL	4802	1227	3.91
SQLite	6959	3211	2.17
	MAXTEST = 256		
MySQL	5495	6199	0.88
PostgreSQL	8554	2852	3.00
SQLite	12900	6529	1.98

7 Enlarging the database

In each of the tests so far, the database size is well below that of main memory, so the SQL daemons, or the operating system, could potentially use memory-mapped I/O to move all of the data into memory. MySQL seems to do just that, as shown by this fragment of the report from the Unix `top` utility:

```
% top -U mysql
last pid: 17084;  load avg:  1.05, 3.59, 19.0; up 22+00:50:05 14:14:55
445 processes: 1 running, 444 sleeping
CPU states: 13.1% user, 0.0% nice, 1.0% system, 86.0% idle, 0.0% iowait
Kernel: 1189 ctxsw, 1026 intr
Memory: 11G used, 4536M free, 732M buffers, 7182M cached
Swap: 32M used, 31G free, 9796K cached

 PID USERNAME THR PRI NICE SIZE   RES   SHR STATE   TIME    CPU COMMAND
2987 mysql     9  18    0  597M   98M 4228K sleep 150:07  0.00% mysqld
```

Notice that the server has 9 threads running, and is using about 0.6GB of RAM, of which only 0.1GB is currently resident

The `lsof` utility exposes details of the total memory use:

```
% lsof -p 2987
COMMAND  PID  USER   FD   TYPE  DEVICE      SIZE     NODE NAME
mysqld  2987 mysql  cwd    DIR     9,0      4096 22282244 /fusionio/mysql/var/lib/mysql
mysqld  2987 mysql  rtd    DIR     8,1      4096        2 /
mysqld  2987 mysql  txt    REG     8,1   6613619  5047885 /home/local/libexec/mysqld
mysqld  2987 mysql  mem    REG     8,1    139504 13762571 /lib64/ld-2.5.so
mysqld  2987 mysql  mem    REG     8,1   1722304 13762573 /lib64/libc-2.5.so
mysqld  2987 mysql  mem    REG     8,1    615136 13762607 /lib64/libm-2.5.so
mysqld  2987 mysql  mem    REG     8,1     23360 13762605 /lib64/libdl-2.5.so
mysqld  2987 mysql  mem    REG     8,1    145824 13762599 /lib64/libpthread-2.5.so
mysqld  2987 mysql  mem    REG     8,1     53448 13762600 /lib64/librt-2.5.so
mysqld  2987 mysql  mem    REG     8,1    114352 13762604 /lib64/libnsl-2.5.so
mysqld  2987 mysql  mem    REG     8,1     48600 13762622 /lib64/libcrypt-2.5.so
mysqld  2987 mysql  mem    REG     8,1     58400 13762750 /lib64/libgcc_s-4.1.2-20080825.so.1
mysqld  2987 mysql  mem    REG     8,1     43040 16649675 /lib64/libnss_compat-2.5.so
mysqld  2987 mysql  mem    REG     8,1     53432 16649683 /lib64/libnss_nis-2.5.so
mysqld  2987 mysql  mem    REG     8,1     53880 16649679 /lib64/libnss_files-2.5.so
mysqld  2987 mysql  mem    REG     8,1    102982 14060840 /home/local/lib64/libz.so.1.2.5
mysqld  2987 mysql  mem    REG     8,1   4488456  5448910 /home/local/lib64/libstdc++.so.5.0.7
mysqld  2987 mysql    0r   CHR     1,3               1832 /dev/null
mysqld  2987 mysql    1w   REG     8,1      5161 26575167 /var/log/mysqld.log
mysqld  2987 mysql    2w   REG     8,1      5161 26575167 /var/log/mysqld.log
mysqld  2987 mysql    3uW  REG     9,0  10485760 22282245 /fusionio/mysql/var/lib/mysql/ibdata1
mysqld  2987 mysql    4u   REG     8,1         0 25034762 /tmp/ibgYByjI (deleted)
mysqld  2987 mysql    5u   REG     8,1         0 25034763 /tmp/ib2a313f (deleted)
mysqld  2987 mysql    6u   REG     8,1         0 25034764 /tmp/ibJsSvON (deleted)
mysqld  2987 mysql    7u   REG     8,1         0 25034768 /tmp/ibHSzrzl (deleted)
mysqld  2987 mysql    8uW  REG     9,0   5242880 22282310 /fusionio/mysql/var/lib/mysql/ib_logfile0
mysqld  2987 mysql    9uW  REG     9,0   5242880 22282298 /fusionio/mysql/var/lib/mysql/ib_logfile1
mysqld  2987 mysql   10u  IPv4 6559078                TCP *:mysql (LISTEN)
mysqld  2987 mysql   11u   REG     8,1         0 25034769 /tmp/ibf9mrlT (deleted)
mysqld  2987 mysql   12u  unix     ...            6559079 /var/lib/mysql/mysql.sock
mysqld  2987 mysql   13u   REG     9,0 694737740 22282302 /fusionio/mysql/var/lib/mysql/bibtex/bibtab.MYD
mysqld  2987 mysql   15u   REG     9,0 694737740 22282302 /fusionio/mysql/var/lib/mysql/bibtex/bibtab.MYD
...
mysqld  2987 mysql  256u   REG     9,0 694737740 22282302 /fusionio/mysql/var/lib/mysql/bibtex/bibtab.MYD
mysqld  2987 mysql  260u   REG     9,0 694737740 22282302 /fusionio/mysql/var/lib/mysql/bibtex/bibtab.MYD
```

The many similar lines at the end show that the 694MB `bibtab.MYD` database file has been mapped into memory, with about 250 file handles pointing at parts of the data.

Similar measurements during the PostgreSQL benchmark shows that its daemon forks one single-threaded copy of itself for each incoming connection, contributing to the overhead compared to MySQL's multithreaded single daemon. Each copy has only one or two open files. A single PostgreSQL database consists of many separate files: there are 1047 files for the BIBTEX database in these benchmarks. That too supplies additional overhead compared to MySQL.

SQLite shows quite different characteristics:

```
% top
...
  PID USERNAME  THR PRI NICE  SIZE   RES   SHR STATE   TIME    CPU COMMAND
11707 beebe       1  18    0   26M 4388K 1344K disk    1:01  1.80% sqlite3
11233 beebe       1  18    0   26M 4384K 1344K run     1:03  1.60% sqlite3
11848 beebe       1  18    0   26M 4388K 1344K disk    1:03  1.60% sqlite3
...
```

Each of its single-threaded processes is only 26MB, and there is only one open database file with a 4MB block loaded into memory.

To better show the throughput improvement possible with solid-state storage compared to magnetic disk, it is necessary to increase the database size beyond the RAM available to the CPU.

In this particular application, it is difficult to increase the database size with real data: the TUG collection so far represents about 15 years of intermittent human effort.

We could increase the database size arbitrarily by replicating and randomizing the output of `bibtosql`, so that new entries added to the database have additional field/value pairs, without disturbing their original contents. However, we have chosen not to do so, because it would require modifying the already-existing, and in-use, disk-based databases, or else replicating them to new servers before expanding them. The required effort was more than this human benchmarker cared to undertake.

8 Summary and conclusions

Benchmarking and analyzing performance of real computer systems is difficult, because there are so many variables over which one has little control, including filesystem design, O/S and database software design, network connections, and kernel device drivers.

The results discussed and tabulated in this report show that there is great variability in performance between different databases with almost identical contents and similar search queries. When the contents and queries are held constant, and only the source of the data — disk or solid-state storage — is changed, then the Fusion-io storage device is a good choice for PostgreSQL and SQLite, but makes little difference for MySQL, most likely because of its use of memory-mapped I/O.

The results show that databases need to be benchmarked against one another before a commitment is made to one of them for a real-world application. It is important to include other kinds of benchmarking too — our results reflect only *read* performance, not *create* or *write* performance, both of which are likely to be important in transactional databases.

It is imperative to understand the ACID (Atomicity, Consistency, Isolation, and Durability) model of databases, because not all of them can provide those guarantees in all circumstances. A failure of cooling, electrical power, hardware, networking, or software that leads to database corruption is likely to be extremely costly in a real-world application, particularly if it occurs during peak client demands, when server hardware is most vulnerable.

Database backup is also important [4, 6], but doing so may require shutting down the database server for extended periods in order to provide a consistent stable disk image for backup software. Some advanced filesystems provide a snapshot capability, so that can be used after a database *flush* operation to freeze the state of the filesystem within a few seconds. The faster the I/O system that holds the database, and the smaller the database storage is, the sooner client services can be restored. We saw that the same input data required from

1.1GB to 3.5GB, depending on the database. Changing databases could dramatically change the cost of online storage, backup storage, and backup time.

References

[1] Nelson H. F. Beebe. BIBTEX meets relational databases: Dedicated to the memory of Edgar Frank "Ted" Codd (1923–2003) and James Nicholas "Jim" Gray (1944–2007). *TUGboat*, 30(2):252–271, 2009. URL `http://www.tug.org/TUGboat/tb30-2/tb95beebe.pdf`. TUG 2009 Conference Proceedings volume.

[2] Sibsankar Haldar. *Inside SQLite*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2007. ISBN 0-596-55006-5. LCCN QA76.73.S67 H35 2007eb; QA76.73.S67. URL `http://www.oreilly.com/catalog/9780596550066`.

[3] Sanjay Mishra and Alan Beaulieu. *Mastering Oracle SQL*. O'Reilly Media, Inc., Sebastopol, CA, USA, second edition, 2004. ISBN 0-596-00632-2. xvii + 472 pp. LCCN QA76.9.D3 M5787 2004. URL `http://www.oreilly.com/catalog/9780596006327`.

[4] W. Curtis Preston. *Backup and recovery*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2007. ISBN 0-596-10246-1 (paperback). xxviii + 729 pp. LCCN QA76.9.B32 P74 2007. URL `http://www.oreilly.com/catalog/9780596102463`.

[5] John C. Worsley and Joshua D. Drake. *Practical PostgreSQL*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002. ISBN 1-56592-846-6. xiv + 619 pp. LCCN QA76.9.D3 W67 2002; QA76.9.D3 W72 2002. US$44.95. URL `http://www.oreilly.com/catalog/9781565928466`. CD-ROM contains LXP version 0.8.0 and PostgreSQL version 7.1.3.

[6] Jeremy D. Zawodny and Derek J. Balling. *High performance MySQL: optimization, backups, replication, and load balancing*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004. ISBN 0-596-00306-4. xvi + 276 pp. LCCN QA76.73.S67 Z39 2004. URL `http://www.oreilly.com/catalog/9780596003067`.