VE482 Lab Report

Lab 4 - Fall 2021

Name: Mingxuan Lu
ID: 518021911166
Email: mingxuan.lu@sjtu.edu.cn

Database

Database system installation

- What are the most common database systems?

  MySQL, Redis, Hive

- Briefly list the pros and cons of the three most common ones.
  - MySQL
    - Pros: high performance for large databases, open source
    - Cons: incremental backups are uneasy to implement, no support for XML or OLAP
  - Oracle
    - Pros: Uses SQL language; Great performance to deal with lots of data; Good and vast documentation and community
    - Cons: Compared to the most modern databases, it is hard to learn; Expensive licenses
  - Redis
    - Pros: high speed, easy configuration, multiple data types supported
    - Cons: requires more memory, no support for join queries

- Create an empty SQLite database

```
sqlite3 l4.db
```

- Use the SQLite shell to prepare two empty tables for each of your .csv file.

```
CREATE TABLE db
(
  hash TEXT NOT NULL,
  name TEXT NOT NULL,
  subject TEXT
);
CREATE TABLE time_stamp
(
  hash TEXT NOT NULL,
  name TEXT NOT NULL,
  dates TEXT,
  tstamp INT
);
```

- Import each .csv file in its corresponding SQLite table.

```
.separator "|"
.import db.psv db
.import timestamp.psv time_stamp
```

Database queries

- Who are the top five contributors to the Linux kernel since the beginning?

```
select name, count(*) from time_stamp group by name order by count(*) desc limit 5;
```

We can get the result:

```
Linus Torvalds|30702
David S. Miller|13180
Takashi Iwai|7726
Mark Brown|7670
Arnd Bergmann|7520
```

- Who are the top five contributors to the Linux kernel for each year over the past five years?

```
select name, count(*) from time_stamp where tstamp >= 1451606400 AND tstamp < 1483228800
group by name order by count(*) desc limit 5; #2016
select name, count(*) from time_stamp where tstamp >= 1483228800 AND tstamp < 1514764800
group by name order by count(*) desc limit 5; #2017
select name, count(*) from time_stamp where tstamp >= 1514764800 AND tstamp < 1546300800
group by name order by count(*) desc limit 5; #2018
select name, count(*) from time_stamp where tstamp >= 1546300800 AND tstamp < 1577836800
group by name order by count(*) desc limit 5; #2019
select name, count(*) from time_stamp where tstamp >= 1577836800 AND tstamp < 1609459200
group by name order by count(*) desc limit 5; #2020
```

We can get the result:

```
# 2016
Linus Torvalds|2273
Arnd Bergmann|1185
David S. Miller|1150
Chris Wilson|992
Mauro Carvalho Chehab|975

# 2017
Linus Torvalds|2303
David S. Miller|1420
Arnd Bergmann|1123
Chris Wilson|1028
Arvind Yadav|827

# 2018
Linus Torvalds|2168
David S. Miller|1405
Arnd Bergmann|922
Christoph Hellwig|818
Colin Ian King|798
```

```
# 2019
Linus Torvalds|2386
David S. Miller|1206
Chris Wilson|1173
YueHaibing|930
Christoph Hellwig|911

# 2020
Linus Torvalds|1886
David S. Miller|924
Christoph Hellwig|806
Mauro Carvalho Chehab|770
Chris Wilson|644
```

- What is the most common “commit subject”?

```
select subject, count(*) from db group by subject order by count(*) desc limit 1;
```

We can get the result:

```
Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net|670
```

- On which day is the number of commits the highest?

```
select date(dates), count(*) from time_stamp group by date(dates) order by count(*) desc
limit 1;
```

We can get the result:

```
2008-01-30|1031
```

- Determine the average time between two commits for the five main contributor.

```
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(*) - 1) FROM time_stamp WHERE name = "Linus
Torvalds";
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(*) - 1) FROM time_stamp WHERE name = "David S.
Miller";
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(*) - 1) FROM time_stamp WHERE name = "Takashi
Iwai";
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(*) - 1) FROM time_stamp WHERE name = "Mark
Brown";
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(*) - 1) FROM time_stamp WHERE name = "Arnd
Bergmann";
```

We can get the result:

```
# Linus Torvalds
15881
# David S. Miller
36956
# Takashi Iwai
63301
# Mark Brown
59933
# Arnd Bergmann
63807
```

Debugging

- How to enable built-in debugging in gcc?

  Add `-g` to enable the extra debugging options in gcc

- **What is the meaning of GDB?**

  GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

Basic GDB usage

- Find the homepage of the GDB project.

  https://www.gnu.org/software/gdb/

- What languages are supported by GDB?

  - Ada
  - Assembly
  - C
  - C++
  - D
  - Fortran
  - Go
  - Objective-C
  - OpenCL
  - Modula-2
  - Pascal
  - Rust

- What are the following GDB commands doing

  - `backtrace` : A backtrace is a summary of how your program got where it is. It shows one line per frame, for many frames, starting with the currently executing frame (frame zero), followed by its caller (frame one), and on up the stack.
  - `where` : additional alias for `backtrace` .
  - `finish` : continue execution of the current function until it returns to its caller.
  - `delete` : delete a specified breakpoint
  - `info breakpoints` : displays information about all breakpoints. When one or more numbers are specified, only the breakpoints matching the specified numbers will be displayed.

- Search the documentation and explain how to use conditional breakpoints.

  `condition bnum expression` , where `bnum` is the specified breakpoint number and `expression` is a specified expression.

- What is `-tui` option for GDB?

  use gdb **Text User Interface**

- What is the "reverse step" in GDB and how to enable it. Provide the key steps and commands.

When you are debugging a program, it is not unusual to realize that you have gone too far, and some event of interest has already happened. If the target environment supports it, GDB can allow you to "rewind" the program by running it backward. Take the following steps as an example:

```
(gdb) continue
Continuing.

No more reverse-execution history.
0x005037a0 in _dl_sysinfo_int80 () from /lib/ld-linux.so.2
(gdb) reverse-continue
Continuing.

Breakpoint 3, bar ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:22
22          xyz = 2; /* break in bar */
(gdb) reverse-continue
Continuing.

Breakpoint 2, foo ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:28
28          xyz = 1; /* break in foo */
(gdb) reverse-continue
Continuing.

No more reverse-execution history.
main ()
    at /data/home/msnyder/cvs/localhost/dumpload/gdb/testsuite/gdb.reverse/break-
reverse.c:34
34          xyz = 0;        /* break in main */
(gdb)
```