VE482 — Introduction to Operating Systems — Lab 4

1 Database

1.1 Database creation

1.2 Database system installation

- What are the most common database systems?

  MySQL, MariaDB, MongoDB, Redis, PostgreSQL

- Briefly list the **pros** and **cons** of the three most common ones.
  - MySQL
    - pro: Small size, fast speed, low total cost of ownership, open source; Support multiple operating systems; Is an open source database, provides an interface to support multiple language connection operations;
    - con: Does not support hot backup; The biggest drawback of MySQL is its security system, which is mainly complex rather than standard, and only changes when the user permissions are reread by calling mysqladmin;
  - MariaDB
    - pro: MariaDB is optimized for performance and is much more powerful than MySQL for large data sets. The ability to gracefully migrate to MariaDB from other database systems is another benefit. Switching from MySQL to MariaDB is relatively easy, which is like cake for a system administrator.

        - con: MariaDB cannot be migrated back to MySQL starting from version 5.5.36. For a new version of MariaDB, the corresponding library (for Debian) will not be deployed in time, which will result in the need to upgrade to a newer version due to dependencies. The clustered version of MariaDB is not very stable.
    - MongoDB
        - pro: weak consistency (final consistency), better to ensure user access speed. The storage mode of document structure can make it more convenient to obtain data. For a hierarchical data structure, it can be difficult to query or retrieve data if a lot of data is stored in a flat, tabular structure.
        - con: MongoDB does not support transaction operations, so systems with strict transaction requirements, such as banking systems, cannot use it. Pre-allocation of space: when MongoDB has insufficient space, it will apply to generate a large block of disk space, and the requested space is always 64M, 128M, 256M to increase up to 2G to the larger size of a single file, and as the number increases, you can see in the data directory as a block of files generated and increasing.

After completing your reading you decide to install SQLite on your Linux system. The next step is now to import your git database into two tables.

- Create an empty SQLite database.
- Use the SQLite shell to prepare two empty tables for each of your .csv file.
- Import each .csv file in its corresponding SQLite table.

```
sqlite3 lab4.db

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
);

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

1.3 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:

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

Therefore, the top five contributors to the Linux kernel since the beginning are: Linus Torvalds, David S. Miller, Takashi Iwai, Mark Brown, Arnd Bergmann.

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
    # the timestamp needs to be modify according to the specific year
    tstamp >= 1577836800 AND tstamp < 1609459200
GROUP BY
    name
ORDER BY
    count(*) DESC
LIMIT 5;
```

We can get:

```
# 2016
# tstamp >= 1451606400 AND tstamp < 1483228800
Linus Torvalds|2273
Arnd Bergmann|1185
David S. Miller|1150
Chris Wilson|992
Mauro Carvalho Chehab|975

# 2017
# tstamp >= 1483228800 AND tstamp < 1514764800
Linus Torvalds|2303
```

```
David S. Miller|1420
Arnd Bergmann|1123
Chris Wilson|1028
Arvind Yadav|827

# 2018
# tstamp >= 1514764800 AND tstamp < 1546300800
Linus Torvalds|2168
David S. Miller|1405
Arnd Bergmann|922
Christoph Hellwig|818
Colin Ian King|798

# 2019
# tstamp >= 1546300800 AND tstamp < 1577836800
Linus Torvalds|2386
David S. Miller|1206
Chris Wilson|1173
YueHaibing|930
Christoph Hellwig|911

# 2020
# tstamp >= 1577836800 AND tstamp < 1609459200
Linus Torvalds|1886
David S. Miller|924
Christoph Hellwig|806
Mauro Carvalho Chehab|770
Chris Wilson|644
```

Therefore, the top five contributors to the Linux kernel for each year over the past five years:

- 2016: Linus Torvalds, Arnd Bergmann, David S. Miller, Chris Wilson, Mauro Carvalho Chehab
- 2017: Linus Torvalds, David S. Miller, Arnd Bergmann, Chris Wilson, Arvind Yadav
- 2018: Linus Torvalds, David S. Miller, Arnd Bergmann, Christoph Hellwig, Colin Ian King
- 2019: Linus Torvalds, David S. Miller, Chris Wilson, YueHaibing, Christoph Hellwig
- 2020: Linus Torvalds, David S. Miller, Christoph Hellwig, Mauro Carvalho Chehab, Chris Wilson

What is the most common “commit subject”?

```
SELECT
    subject, count(*)
FROM
    db
GROUP BY
    subject
ORDER BY
    count(*) DESC
LIMIT 5;
```

We can get:

```
Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net|670
Merge branch 'for-linus' of
git://git.kernel.org/pub/scm/linux/kernel/git/dtor/input|301
Merge branch 'x86-urgent-for-linus' of
git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip|275
Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net-2.6|262
Merge branch 'perf-urgent-for-linus' of
git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip|248
```

Therefore, the most common "commit subject" is `Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net`

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 5;
```

We can get:

```
2008-01-30|1031
2006-12-07|683
2007-05-08|649
2013-07-03|626
2007-10-16|613
```

Therefore, `2008-01-30` is the number of commits the highest.

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

```
SELECT
    (MAX(tstamp) - MIN(tstamp)) / (count(*) - 1)
FROM
    time_stamp
WHERE
    name = "Arnd Bergmann";
```

We can get:

```
# 487552654 30701
15881
# 487045012 13179
36956
# 489001082 7725
63301
# 459628018 7669
59933
# 479764856 7519
63807
```

2 Debugging

How to enable built-in debugging in `gcc`?

`gcc -g srcfile.c -o output`

What is the meaning of GDB?

GDB is a powerful program debugging tool under Linux released by GNU Open Source Organization. Generally speaking, GDB mainly helps you to complete the following four aspects of the function :

1. Start your application, you can according to your custom requirements to run the application as you want.
2. You can make the program to be debugged at the set breakpoint you specify to stop. (A breakpoint can be a conditional expression.)
3. When the program is stopped, check what is happening in your program.
4. You can test other bugs by changing your program to correct the effects of one BUG

Compile the master branch of you `mumsh` with debugging enabled.

`gcc -g *.c -o mumsh`

2.1 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`: Can be used to trace the function call stack. For example, when a segment error occurs, `backtrace` can be executed to see where the call caused the segment error.
- `where`: same as `hacktrace`
- `finish`: If you have entered a function and want to exit the function and return to its calling function, use the command `finish`.
- `delete`: Delete a breakpoint or monitor; It can also be used with other commands.
- `info breakpoints`: Used to display defined breakpoints.

Search the documentation and explain how to use conditional breakpoints.

To set a conditional breakpoint, use the break if command, as shown below:

```
(gdb) break line-or-function if expr
```

Ex. :

```
(gdb) break 46 if testsize==100
```

What is -tui option for GDB?

TUI (Text User Interface) is a Text-User Interface for GDB debugging, which can easily display source code, assembly and register text Windows

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

Assuming you are using one of the gdb targets that supports reverse debugging (such as Process Record for Linux), you can use the following commands:

- reverse-continue

Run the program backward until it hits a stop event (such as a breakpoint, watchpoint, or exception).

- reverse-step

Run the program backward until the beginning of the previously-executed source line.

- reverse-stepi

Run the program backward for exactly one machine instruction.

- reverse-next

Like "next" in reverse -- function calls are stepped over instead of into (in reverse).

- reverse-nexti

Like "nexti" in reverse -- executes one machine instruction (backward), unles that instruction is a return from a function call, in which case the entire function will be executed in reverse.

- reverse-finish

By analogy to the "finish" command, "reverse-finish" executes the current function in reverse, until the execution reaches the calling function, then stops.

- set exec-direction [forward | reverse]

A modal command: when exec-direction is set to "reverse", all ordinary execution commands such as "step" and "continue" will cause the program being debugged to run in reverse.