VE482 - Lab 4

Introduction to Operating Systems

Kexuan Huang 518370910126

October 15, 2021

1 Database

1.1 Database creation

- timestamp.csv

```
git log --pretty="%H,%aN,%aI,%at" > timestamp.csv
```

- db.csv

```
git log --pretty="%H,%aN,%s" > db.csv
```

1.2 Database system installation

What are the most common database systems?

1. Oracle Database
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL
5. MongoDB

Briefly list the pros and cons of the three most common ones.

1. Oracle database

Pros: Access to all major product and technology releases which can provide solutions covering almost every usage scenario.

Cons: Expensive licenses and steep learning curve.

2. MySQL

Pros: Cross-platform support for different OS; Free; Multi-language support.

Cons: Pool documentation and lacking technological support; Performance-poor scaling and requires higher memory.

3. Microsoft SQL Server

Pros: Intergrate with Microsoft products like Visual Studio; Flexible in terms of configurations; Multiple options for Data Security.

Cons: Expensive licenses; More RAM and CPU power is required for working with large amounts of data.

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.csv db
.import timestamp.csv timestamp
```

1.3 Database queries

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

Linus Torvalds, David S. Miller, Takashi Iwai, Mark Brown, Arnd Bergmann.

Input:

```
SELECT name, count(name) AS times FROM time_stamp
GROUP BY name
ORDER BY times DESC
LIMIT 5;
```

Output:

```
name               count(name)
---------------    -----------
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?

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

Input:

```sql
-- 2016 --
SELECT name, count(name) AS times FROM time_stamp
WHERE dates BETWEEN '2016-01-01' AND '2016-12-31'
GROUP BY name
ORDER BY times DESC
LIMIT 5;
-- 2017 --
SELECT name, count(name) AS times FROM time_stamp
WHERE dates BETWEEN '2017-01-01' AND '2017-12-31'
GROUP BY name
ORDER BY times DESC
LIMIT 5;
-- 2018 --
SELECT name, count(name) AS times FROM time_stamp
WHERE dates BETWEEN '2018-01-01' AND '2018-12-31'
GROUP BY name
ORDER BY times DESC
LIMIT 5;
-- 2019 --
SELECT name, count(name) AS times FROM time_stamp
WHERE dates BETWEEN '2019-01-01' AND '2019-12-31'
GROUP BY name
ORDER BY times DESC
LIMIT 5;
-- 2020 --
SELECT name, count(name) AS times FROM time_stamp
WHERE dates BETWEEN '2020-01-01' AND '2020-12-31'
GROUP BY name
ORDER BY times DESC
LIMIT 5;
```

Output:

```
-- 2016 --
name                   times
---------------------  -----
Linus Torvalds         2273
Arnd Bergmann          1185
David S. Miller        1150
Chris Wilson           988
Mauro Carvalho Chehab  975

-- 2017 --
name             times
---------------  -----
Linus Torvalds   2288
David S. Miller  1420
Arnd Bergmann    1123
Chris Wilson     1028
Arvind Yadav     827

-- 2018 --
name               times
-----------------  -----
Linus Torvalds     2163
David S. Miller    1405
Arnd Bergmann      919
Christoph Hellwig  818
Colin Ian King     798

-- 2019 --
name               times
-----------------  -----
Linus Torvalds     2380
David S. Miller    1205
Chris Wilson       1170
YueHaibing         929
Christoph Hellwig  911

-- 2020 --
name                   times
---------------------  -----
Linus Torvalds         1886
David S. Miller        923
Christoph Hellwig      806
Mauro Carvalho Chehab  770
Chris Wilson           644
```

What is the most common “commit subject”?

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

Input:

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

Output:

```
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
```

On which day is the number of commits the highest?

2008-01-30

Input:

```sql
SELECT date(dates) AS date, count(dates) AS times FROM time_stamp
GROUP BY times
ORDER BY times DESC
LIMIT 5;
```

Output:

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

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

Linus Torvalds: 15880
David S. Miller: 36956
Takashi Iwai: 63301
Mark Brown: 59933
Arnd Bergmann: 63807

Input:

```sql
-- Linus Torvalds --
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(name) - 1) FROM time_stamp
WHERE name = "Linus Torvalds";

-- David S. Miller --
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(name) - 1) FROM time_stamp
WHERE name = "David S. Miller";

-- Takashi Iwai --
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(name) - 1) FROM time_stamp
WHERE name = "Takashi Iwai";

-- Mark Brown --
SELECT (MAX(tstamp) - MIN(tstamp)) / (count(name) - 1) FROM time_stamp
WHERE name = "Mark Brown";

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

Output:

```
-- Linus Torvalds --
15880
-- David S. Miller --
36956
-- Takashi Iwai --
63301
-- Mark Brown --
59933
-- Arnd Bergmann --
63807
```

2 Debugging

1. How to enable built-in debugging in gcc?

> To tell GCC to emit extra information for use by a debugger, in almost all cases you need only to add -g to your other options.

2. 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.
>
> GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:
>
> - Start your program, specifying anything that might affect its behavior.
> - Make your program stop on specified conditions.
> - Examine what has happened, when your program has stopped.
> - Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

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

```sh
#!/bin/sh
# Compile on MacOS
sudo killall taskgated
codesign -fs gdb-cert "$(which gdb)"
set startup-with-shell off
echo "set startup-with-shell off" >> ~/.gdbinit
clang *.c -o mumsh_debug -ggdb
```

2.1 Basic GDB usage

1. Find the homepage of the GDB project.

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

2. What languages are supported by GDB?

> GDB supports the following languages (in alphabetical order):
>
> Ada, Assembly, C, C++, D, Fortran, Go, Objective-C, OpenCL, Modula-2, Pascal, Rust

3. What are the following GDB commands doing:

- `backtrace`: A summary of how your program got where it is
- `where`: Additional aliases for `backtrace`.
- `finish`: Continue running until just after function in the selected stack frame returns.
- `delete`: Permanently delete one or more tracepoints.
- `info breakpoints`: Print a table of all breakpoints, watchpoints, and catchpoints set and not deleted.

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

> Command format: `condition bnum expression`
>
> Specify expression as the break condition for breakpoint, watchpoint, or catchpoint number bnum. After you set a condition, breakpoint bnum stops your program only if the value of expression is true (nonzero, in C).

For example, bnum can be "`1`" and expression can be "`i>482`".

5. What is `-tui` option for GDB?

Option `-tui` means use "Text User Interface", which can also be activated by pressing `CTRL-XA`.

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

> GDB can allow you to "rewind" the program by running it backward.

1. Set two breakpoints 2 and 3
2. Set the rule for breakpoints 2 and 3

```
command 3\n run\n end
command 2\n record\n continue\n end
```

3. (Optional) Diable pagination display for tidiness: `set pagination off`
4. Rerun and stop when a bug occurs: `run`
5. Reverse-execute one machine instruction: `reverse-stepi`