Student: Zhang Liqin

UID: 517370910123

VE482 Lab4

Database

Database creation

According to the git pretty format, we use the following placeholders to represent the required fields.

timestamp.csv

- Hash of the commit - `%H`
- Author name - `%an`
- Author date, strict ISO 8601 format - `%aI`
- Author date, UNIX timestamp - `%at`

Or we could use the following code

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

db.csv

- Hash of the commit - `%H`
- Author name - `%an`
- Subject - `%s`

Similarly, we could use the following code

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

Database system installation

- What are the most common database systems?

    - Oracle 12c, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB...
- Briefly list the pros anc cons of the three most common ones.
    1. MongoDB:

       Pros:

        - It's schema-less
        - It's easy to scale-out

       Cons:

        - High data size since document has stored field names
        - Less flexibilty with querying
        - No support for transactions
    2. MySQL

       Pros:

        - Storing relational data
        - Quick management
        - Open source eco-system

       Cons:

        - Restriction on table sizes and schemes
        - Licensing from Oracle
    3. Oracle 12c

       Pros:

        - Able to scale to support intensive workloads
        - Provide efficient ways to developing data-intensive process
        - Partitioning, compression and encryption are optional features

       Cons:

        - Too new, multi-tenant architecture isn't so straightforward as SQL
        - Requires addtional licensing and thus expensive
- 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 timestamp
(
  hash text NOT NULL,
  name text NOT NULL,
  time_iso text NOT NULL,
  time_unix text NOT NULL
);

CREATE TABLE db
(
  hash text NOT NULL,
  name text NOT NULL,
  subject text NOT NULL
);
```

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

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

Database queries

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

```
select name, cnt from (select name, COUNT(name) as cnt from db
group by name) order by cnt DESC limit 5;

/***** Output *****/
Linus Torvalds,30664
David S. Miller,13166
Takashi Iwai,7722
Mark Brown,7660
Arnd Bergmann,7510
```

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

```
select name, COUNT(name) as cnt from (select * from timestamp as t
where strftime('%Y',t.time_iso)='2020') group by name order by cnt
DESC limit 5;
/* Repeat the similar for year 2016-2019*/
/***** Output *****/
/*2020*/
Linus Torvalds,1886
David S. Miller,924
```

```
Christoph Hellwig,806
Mauro Carvalho Chehab,770
Chris Wilson,644

/*2019*/
Linus Torvalds,2386
David S. Miller,1206
Chris Wilson,1173
YueHaibing,930
Christoph Hellwig,911

/*2018*/
Linus Torvalds,2168
David S. Miller,1405
Arnd Bergmann,922
Christoph Hellwig,818
Colin Ian King,798

/*2017*/
Linus Torvalds,2303
David S. Miller,1420
Arnd Bergmann,1123
Chris Wilson,1028
Arvind Yadav,827

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

- What is the most common “commit subject”?

```
select subject, count(subject) as cnt from db group by subject
order by cnt DESC limit 1;
/***** Output *****/
Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net|670
```

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

```
select time_iso, count(hash) as cnt from timestamp as t GROUP BY
strftime ('%Y-%m-%d',t.time_iso) order by cnt DESC limit 1;
/***** Output *****/
2008-01-30T14:39:54-05:00,1013
```

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

Since for average time calculation, we could cancel out all the intermediate time and conclude $\frac{\text{Last}-\text{First}}{\text{Commit times}-1}$.

First, we find the five main contributors.

```
select name, count(hash) as cnt from timestamp group by name order
by cnt DESC limit 5;
/***** Output *****/
Linus Torvalds|30702
David S. Miller|13180
Takashi Iwai|7726
Mark Brown|7670
Arnd Bergmann|7520
```

Then we reformat one's first and latest submisstion time difference in `%s` , which is the second counts from 1970-01-01.

```
select((select strftime('%s',(select max(time_iso) from timestamp
where name=='Linus Torvalds')))-(select strftime('%s',(select
min(time_iso) from timestamp where name=='Linus Torvalds'))));
487552654
select((select strftime('%s',(select max(time_iso) from timestamp
where name=='David S. Miller')))-(select strftime('%s',(select
min(time_iso) from timestamp where name=='David S. Miller'))));
487045012
select((select strftime('%s',(select max(time_iso) from timestamp
where name=='Takashi Iwai')))-(select strftime('%s',(select
min(time_iso) from timestamp where name=='Takashi Iwai'))));
489001082
select((select strftime('%s',(select max(time_iso) from timestamp
where name=='Mark Brown')))-(select strftime('%s',(select
min(time_iso) from timestamp where name=='Mark Brown'))));
459628018
select((select strftime('%s',(select max(time_iso) from timestamp
where name=='Arnd Bergmann')))-(select strftime('%s',(select
min(time_iso) from timestamp where name=='Arnd Bergmann'))));
479764856
```

Using the above results, we can conclude the average time for these five main contributors as:

Linus: $487552654/30701 = 15880.676655484s \approx 4.4h$

David: $487045012/13179 = 36956.143258214s \approx 10.3h$

Takashi: $489001082/7725 = 63301.110938511s \approx 17.6h$

Mark Brown: $459628018/7669 = 59933.240057374s \approx 16.6h$

Arnd Bergmann: $479764856/7519 = 63807.003058917s \approx 17.7h$

Debugging

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

Using option `-g` while compiling. For example,

```
g++ -g hello.cpp -o hello
```

2. What is the meaning of GDB?

GDB is the GNU Project Debugger, which is a powerful debugging tool for C.

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

```
gcc -g -o homework4 homework4.c
```

Basic GDB usage

1. Find the homepage of the GDB project.

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

2. What languages are supported by GDB?

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

3. What are the following GDB commands doing:
    - backtrace: print one line per frame for all frames in the stack a backtrace of the entire stack
    - delete: permanently delete one or more breakpoints
    - where: the synonyms with 'backtrace' and produce the same output
    - info breakpoints: check status of all specified breakpoints
    - finish: execute program unless selected stack frame returns
4. Search the documentation and explain how to use conditional breakpoints.

Conditional breakpoints is a way for user to conditionally stop the program. First, we set a breakpoint in the program

```
(gdb) break <file_name> : <line_number>
(gdb) break <function_name>
```

Then we set a condition for the break point

```
(gdb) condition <breakpoint_number> condition
```

When the condition is no longer needed, we could remove the condition using

```
(gdb) condition <breakpoint_number>
```

5. What is -tui option for GDB?

The GDB Text User Interface (TUI) is a terminal interface which uses the `curses` library to show the source file, the assembly output, the program registers and GDB commands in separate text windows. The TUI mode is supported only on platforms where a suitable version of the `curses` library is available.

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

Reverse step is included in GDB version 7.0, to enabling reverse debugging, one need to save enough machine state that can be restored later. There are a number of schemes, one of which is to simply save the registers or memory locations that are modified by each machine instruction. Then, to "undo" that instruction, one just revert the data in those registers or memory locations. The following are all new commands to support this feature:

- **reverse-continue** ('rc') -- Continue program being debugged but run it in reverse
- **reverse-finish** -- Execute backward until just before the selected stack frame is called
- **reverse-next** ('rn') -- Step program backward, proceeding through subroutine calls.
- **reverse-nexti** ('rni') -- Step backward one instruction, but proceed through called subroutines.
- **reverse-step** ('rs') -- Step program backward until it reaches the beginning of a previous source line
- **reverse-stepi** -- Step backward exactly one instruction
- **set exec-direction (forward/reverse)** -- Set direction of execution. All subsequent execution commands (continue, step, until etc.) will run the program being debugged in the selected direction.