**SQL**

Таблица history находится в прикреплённом файле test.db (SQLite)

*history*

issue\_key – уникальный ключ задачи

status – статус задачи

minutes\_in\_status – количество минут, которое задача находилась в статусе

previous\_status – предыдущий статус задачи

started\_at – время создания статуса задачи, unix миллисекунды

ended\_at – время перехода задачи в другой статус, unix миллисекунды

*SQL 1*

Напишите запрос, который выведет, сколько времени в среднем задачи каждой группы находятся в статусе “Open”

Условия:

Под группой подразумевается первый символ в ключе задачи. Например, для ключа “C-40460” группой будет “C”

Задача может переходить в один и тот же статус несколько раз.

Переведите время в часы с округлением до двух знаков после запятой.

Ответ:

ALTER TABLE history

ADD group\_key varchar2(5);

UPDATE history

SET group\_key = SUBSTR(issue\_key, 1, INSTR(issue\_key, '-')-1),

issue\_key = SUBSTR(issue\_key, INSTR(issue\_key, '-')+1);

-- 1 мин. / 60 мин. = 0.01666666667

SELECT group\_key,

ROUND(AVG((minutes\_in\_status \* 0.01666666667)),2) as avg\_hour\_in\_status

FROM 'history'

where status = 'Open'

group by group\_key;

*SQL 2*

Напишите запрос, который выведет ключ задачи, последний статус и его время создания для задач, которые открыты на данный момент времени.

Условия:

Открытыми считаются задачи, у которых последний статус в момент времени не “Closed” и не “Resolved”

Задача может переходить в один и тот же статус несколько раз.

Оформите запрос таким образом, чтобы, изменив дату, его можно было использовать для поиска открытых задач в любой момент времени в прошлом

Переведите время в текстовое представление

Ответ:

SELECT group\_key, issue\_key, status, max(started\_at),

DATE(ROUND(started\_at/ 1000), 'unixepoch') as date\_

FROM history

WHERE (DATE(ROUND(started\_at/ 1000), 'unixepoch') = '2022-11-25')

GROUP BY group\_key, issue\_key

having status NOT IN ('Closed', 'Resolved')

или

SELECT issue\_key, status, max(started\_at),

DATE(ROUND(started\_at/ 1000), 'unixepoch') as date\_

FROM history

WHERE (DATE(ROUND(started\_at/ 1000), 'unixepoch') = '2022-11-25')

GROUP BY issue\_key

having status NOT IN ('Closed', 'Resolved')

**PYTHON**

## Слияние логов

Имеется два файла с логами в формате **JSONL**, пример лога:

…

{"timestamp": "2021-02-26 08:59:20", "log\_level": "INFO", "message": "Hello"}

{"timestamp": "2021-02-26 09:01:14", "log\_level": "INFO", "message": "Crazy"}

{"timestamp": "2021-02-26 09:03:36", "log\_level": "INFO", "message": "World!"}

…

Сообщения в заданных файлах упорядочены по полю timestamp в порядке возрастания.

Требуется написать скрипт, который объединит эти два файла в один.

При этом сообщения в получившемся файле тоже должны быть упорядочены в порядке возрастания по полю timestamp.

К заданию прилагается вспомогательный скрипт на python3, который создает два файла "log\_a.jsonl" и "log\_b.jsonl".

Командлайн для запуска:

log\_generator.py <path/to/dir>

Ваше приложение должно поддерживать следующий командлайн:

<your\_script>.py <path/to/log1> <path/to/log2> -o <path/to/merged/log>

**log\_generator.py**

|  |
| --- |
| import json  import time import argparse import dataclasses import random import shutil  from datetime import datetime, timedelta from pathlib import Path  \_MAX\_LOG\_SIZE\_BYTES = 2 \*\* 30 # 1GB  \_LOG\_FILENAMES = 'log\_a.jsonl', 'log\_b.jsonl' \_LOG\_LEVELS = b'DEBUG', b'INFO', b'WARNING', b'ERROR'  \_PERSON\_NAME = 'Bender', 'Fry', 'Leela', 'Amy', 'Farnsworth', 'Dr. Zoidberg' \_ACTION = 'said', 'took', 'played', 'ate', 'saw', 'built', 'killed', 'created', 'brought', 'robbed' \_OBJECT = 'an apple', 'a car', 'a boat', 'a rocket', 'a mall', 'a fish', 'a bottle of bear', 'a man' \_PLACE = 'at park', 'on the Mars', 'near the Square Garden', 'in L.A.' \_WHEN = 'day before yesterday', 'yesterday', 'today', 'tomorrow', 'day after tomorrow'   @dataclasses.dataclass class LogRecord:  log\_level: str  timestamp: str  message: str   def \_parse\_args() -> argparse.Namespace:  parser = argparse.ArgumentParser(description='Tool to generate test logs.')   parser.add\_argument(  'output\_dir',  metavar='<OUTPUT DIR>',  type=str,  help='path to dir with generated logs',  )   parser.add\_argument(  '-f', '--force',  action='store\_const',  const=True,  default=False,  help='force write logs',  dest='force\_write',  )   return parser.parse\_args()   def \_create\_dir(dir\_path: Path, \*, force\_write: bool = False) -> None:  if dir\_path.exists():  if not force\_write:  raise FileExistsError(  f'Dir "{dir\_path}" already exists. Remove it first or choose another one.')  shutil.rmtree(dir\_path)   dir\_path.mkdir(parents=True)   \_RECORD\_TEMPLATE = LogRecord(  log\_level='<LOG\_LEVEL>',  timestamp='<TIMESTAMP>',  message='<MESSAGE>', )  \_MESSAGE\_TEMPLATE = json.dumps(dataclasses.asdict(\_RECORD\_TEMPLATE)).encode('utf-8') \_MESSAGE\_TEMPLATE += b'\n'   def \_generate\_logfile(log\_filepath: Path, start\_time: datetime) -> None:  print(f"generating {log\_filepath.name}...")  person\_name, action = \_PERSON\_NAME, \_ACTION  object, place, when = \_OBJECT, \_PLACE, \_WHEN  log\_levels, message\_template = \_LOG\_LEVELS, \_MESSAGE\_TEMPLATE  rand, td, ln = random.random, timedelta, len   with log\_filepath.open('wb') as fh:  current\_time = start\_time  total\_size, max\_size = 0, \_MAX\_LOG\_SIZE\_BYTES  write = fh.write  while total\_size < max\_size:  timestamp = f"{current\_time.year}-{current\_time.month:02}-{current\_time.day:02} " \  f"{current\_time.hour}:{current\_time.minute:02}:{current\_time.second:02}".encode('utf-8')   message = f"{person\_name[int(6 \* rand())]} " \  f"{action[int(10 \* rand())]} " \  f"{object[int(8 \* rand())]} " \  f"{place[int(4 \* rand())]} " \  f"{when[int(5 \* rand())]}".encode('utf-8')   data = message\_template \  .replace(b'<LOG\_LEVEL>', log\_levels[int(4 \* rand())]) \  .replace(b'<TIMESTAMP>', timestamp) \  .replace(b'<MESSAGE>', message)   write(data)  total\_size += ln(data)  current\_time += td(seconds=int(10 \* rand()))   def \_generate\_logs(output\_dir: Path) -> None:  start\_time = datetime.now()   for log\_filename in \_LOG\_FILENAMES:  log\_path = output\_dir.joinpath(log\_filename)  \_generate\_logfile(log\_path, start\_time)   def main() -> None:  args = \_parse\_args()   t0 = time.time()  output\_dir = Path(args.output\_dir)  \_create\_dir(output\_dir, force\_write=args.force\_write)  \_generate\_logs(output\_dir)  print(f"finished in {time.time() - t0:0f} sec")   if \_\_name\_\_ == '\_\_main\_\_':  main() |