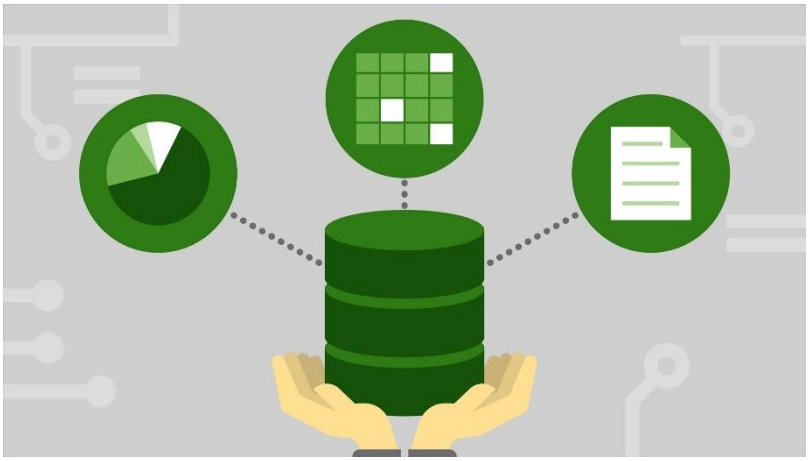
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستور کار شماره ۸

سجاد علی‌زاده
۸۱۰۱۹۷۵۴۷

دی ماه ۱۴۰۰

گزارش دستورکار انجام شده

ابتدا ردیس را نصب کردیم و فایل کانفیگ آن را نیز تولید کردیم. پس از آن با کتابخانه پایتون آن را نیز با استفاده از pip نصب کردیم. حال به اجرای دستورات میپردازیم:

```
>>> import redis
>>> r = redis.Re
redis.ReadOnlyError(  redis.Redis(          redis.RedisError(     redis.ResponseError(
>>> r = redis.Redis()
>>> r.mset({"Croatia": "Zagreb", "Bahamas": "Nassau"})
True
>>> r.get("Bahamas")
b'Nassau'
>>>
```

در این قسمت ابتدا کتابخانه آن را ایمپورت کردیم و یک نمونه از آن ساختیم. سپس یک دیکشنری را در آن اضافه کردیم که موفقیت آمیز بودن آن نیز گزارش شد. سپس مقدار Bahamas آن دریافت شده است.

```
>>> import datetime
>>> today = datetime.date.today()
>>> visitors = {"dan", "jon", "alex"}
>>> r.sadd(today, *visitors)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/sajializ/.local/lib/python3.8/site-packages/redis/commands/c
    return self.execute_command('SADD', name, *values)
  File "/home/sajializ/.local/lib/python3.8/site-packages/redis/client.py"
    return conn.retry.call_with_retry(
  File "/home/sajializ/.local/lib/python3.8/site-packages/redis/retry.py",
    return do()
```

در این قطعه کد یک آبجکت datetime به عنوان کلید انتخاب شده است. اما این کد به ارور میخورد زیرا کلید نمیتواند از این آبجکت باشد. برای حل این مشکل از متد isoformat استفاده میکنیم تا این تاریخ تبدیل به استرینگ شود. یعنی این کد را به شکل زیر بازنویسی میکنیم:

```
>>> stoday = today.isoformat()
>>> stoday
'2021-12-24'
>>> r.sadd(stoday, *visitors)
3
>>> r.smembers(stoday)
{b'jon', b'dan', b'alex'}
>>> r.scard(today.isoformat())
3
>>>
```

ابتدا تاریخ امروز را تبدیل به استرینگ کردیم و در متغیر stoday ریختیم. سپس با این کلید داده‌ها را دریافت کردیم. با دریافت مقادیر آن مشاهده میکنیم به درستی ست شده است.

حال مثال pyhats را اجرا میکنیم.

ابتدا ۳ کلاه با استفاده از دیکشنری پایتون تعریف می‌کنیم:

```
import random

random.seed(444)
hats = {f"hat:{random.getrandbits(32)}": i for i in (
    {
        "color": "black",
        "price": 49.99,
        "style": "fitted",
        "quantity": 1000,
        "npurchased": 0,
    },
    {
        "color": "maroon",
        "price": 59.99,
        "style": "hipster",
        "quantity": 500,
        "npurchased": 0,
    },
    {
        "color": "green",
        "price": 99.99,
        "style": "baseball",
        "quantity": 200,
        "npurchased": 0,
    })
}
```

حال ابتدا این دیکشنری را پرینت میکنیم تا اطلاعات کلاه‌ها را داشته باشیم. سپس یک اینستنس ردیس با شماره دیتابیس ۱ میسازیم (صفر قبلی بود). برای ذخیره این داده‌ها از تایپ hash استفاده میکنیم. برای اینکه همه دستورات یکجا اجرا شوند و atomic باشند از پایپ استفاده میکنیم. چند دستور در پایپ اضافه میشود و با اجرای آن دستورات یکجا فرستاده می‌شود.

```
print(hats)

r = redis.Redis(db=1)
with r.pipeline() as pipe:
    for h_id, hat in hats.items():
        pipe.hmset(h_id, hat)
    pipe.execute()

r.bgsave()
```

خروجی دو قسمت بالا به شکل زیر است:

```
→  DB python3 1.py
{'hat:1326692461': {'color': 'black', 'price': 49.99, 'style': 'fitted', 'quantity': 1000, 'npurchased'
: 0}, 'hat:1236154736': {'color': 'maroon', 'price': 59.99, 'style': 'hipster', 'quantity': 500, 'npurc
hased': 0}, 'hat:56854717': {'color': 'green', 'price': 99.99, 'style': 'baseball', 'quantity': 200, 'n
purchased': 0}}
1.py:33: DeprecationWarning: Pipeline.hmset() is deprecated. Use Pipeline.hset() instead.
  pipe.hmset(h_id, hat)
```

حال ابتدا چک میکنیم کلاه به ردیس اضافه شده است یا خیر. همچنین بعد از آن تمام کلیدها را چاپ میکنیم. میبینیم همه چیز به خوبی پیش رفته است.

```
>>> print(r.hgetall("hat:56854717"))
{b'color': b'green', b'price': b'99.99', b'style': b'baseball', b'quantity': b'200', b'npurchased': b'0
'}
>>>
>>> r.keys()
[b'hat:1326692461', b'hat:56854717', b'hat:1236154736']
```

حال توابع زیر را بررسی میکنیم:

```
>>> r.hincrby("hat:56854717", "quantity", -1)
199
>>> r.hget("hat:56854717", "quantity")
b'199'
>>> r.hincrby("hat:56854717", "npurchased", 1)
1
```

مشاهده میکنیم با استفاده از تابع hincby میتوانیم برای یک آبجکت یک مقدار را به یک اندازه زیاد یا کم کنیم. مثلا در خط اول مقدار quantity به میزان منفی یک اضافه شده و مقدار آن به ۲۰۰ منهای ۱ یعنی ۱۹۹ رسیده است. (با گرفتن آن نیز به همین نتیجه رسیدیم) یا مثلا در خط آخر برای یک آبجکت مقدار npurchased را یکی زیاد کردیم و به یک افزایش دادیم. با کارهایی که تا الان انجام دادیم میتوانیم تابع خرید یک کلاه را بنویسیم. برای این تابع باید در صورتی که موجودی داشته باشیم مقدار npurchased را یکی زیاد کرده و مقدار quantity را یکی کم کنیم. این دو عملیات باید در یک پایپ

انجام شوند تا یا هر دو اجرا شوند یا هیچکدام اجرا نشوند. همچنین نباید موجودی منفی شود که این کار با استفاده از watch انجام می‌شود. کد تابع به شکل زیر است (فایل 2.py):

```
def buyitem(r: redis.Redis, itemid: int) -> None:
    with r.pipeline() as pipe:
        error_count = 0
        while True:
            try:
                # Get available inventory, watching for changes
                # related to this itemid before the transaction
                pipe.watch(itemid)
                nleft: bytes = r.hget(itemid, "quantity")
                if nleft > b"0":
                    pipe.multi()
                    pipe.hincrby(itemid, "quantity", -1)
                    pipe.hincrby(itemid, "npurchased", 1)
                    pipe.execute()
                    break
                else:
                    # Stop watching the itemid and raise to break out
                    pipe.unwatch()
                    raise OutOfStockError(
                        f"Sorry, {itemid} is out of stock!"
                    )
            except redis.WatchError:
                # Log total num. of errors by this user to buy this item,
                # then try the same process again of WATCH/HGET/MULTI/EXEC
                error_count += 1
                logging.warning(
                    "WatchError #%d: %s; retrying",
                    error_count, itemid
                )
    return None
```

همانطور که مشاهده میشود در ابتدا موجودی کلاه مربوطه را گرفتیم و در صورت صفر نبودن با استفاده از پایپ دستورات اضافه کردن به npurchased و کم کردن از quantity را اضافه کردیم. همچنین با استفاده از watch دقت کردیم موجودی منفی نشود. در صورتی که منفی میشد یک استثنا از نوع watch error میگیریم.

حال به تست کدی که زدیم میپردازیم:

```
r = redis.Redis(db=1)
buyitem(r, "hat:56854717")
buyitem(r, "hat:56854717")
buyitem(r, "hat:56854717")
print(r.hmget("hat:56854717", "quantity", "npurchased"))

for _ in range(196):
    buyitem(r, "hat:56854717")
print(r.hmget("hat:56854717", "quantity", "npurchased"))

buyitem(r, "hat:56854717")
```

نتیجه به شکل زیر است:

```
→  DB python3 2.py
[b'196', b'4']
[b'0', b'200']
Traceback (most recent call last):
  File "2.py", line 51, in <module>
    buyitem(r, "hat:56854717")
  File "2.py", line 27, in buyitem
    raise OutOfStockError(
__main__.OutOfStockError: Sorry, hat:56854717 is out of stock!
```

ابتدا سه کلاه خریدیم و سپس مقادیر quantity و npurchased را چاپ کردیم. مشاهده میکنیم که این مقادیر به درستی آپدیت شده‌اند (توجه کنید در قسمت قبل یک کلاه خریده بودیم) سپس باقیمانده کلاه‌ها را میخریم و مشاهده میکنیم این مقادیر باز هم به درستی آپدیت شده‌اند. در نهایت نیز یک کلاه دیگر میخریم ولی از آنجایی که موجودی صفر است یک استثنا پرتاب شد.

با دستور setex میتوان برای کلیدها یک expire time تعیین کرد.

```
>>> from datetime import timedelta
>>> import redis
>>> r = redis.Redis(db=2)
>>> r.setex("runner", timedelta(minutes=1), value="salam")
True
>>> r.ttl("runner")
52
>>> r.pttl("runner")
47649
>>> r.get("runner")
b'salam'
>>> r.expire("runner", timedelta(seconds=3))
True
>>> r.get("runner")
>>> r.exists("runner")
0
```

در این کد ابتدا یک کلید به نام runner با زمان انقضای یک دقیقه تولید کردیم. سپس با استفاده از متد ttl زمان باقیمانده آن کلید را برحسب ثانیه و با استفاده از متد pttl زمان را بر حسب میلی ثانیه به دست آوردیم. همچنین یکبار آن را get کردیم تا مطمئن شویم مشکلی وجود ندارد. سپس زمان انقضای آن را به ۳ ثانیه کاهش دادیم و ۳ ثانیه صبر کردیم تا این کلید از بین برود. همانطور که مشاهده میشود بعد از این زمان موقع get کردن چیزی برگردانده نمی‌شود و همچنین تابع exists به ما میگوید همچنین کلیدی وجود ندارد.

از این قابلیت استفاده میکنیم تا آیپی‌هایی که بیش از حد ریکوئست میزنند را بیابیم. به شکل زیر عمل میکنیم:

```
>>> r = redis.Redis(db=5)
>>> r.lpush("ips", "51.218.112.236")
1
>>> r.lpush("ips", "90.213.45.98")
2
>>> r.lpush("ips", "115.215.230.176")
3
>>> r.lpush("ips", "51.218.112.236")
4
```

در یک دیتابیس جدید یک لیست با کلید ips تعریف می‌کنیم و با استفاده از lpush چهار داده به آن اضافه می‌کنیم. حال به پیاده‌سازی کد اصلی می‌پردازیم (فایل 3.py):

```python
import datetime
import ipaddress

import redis

# Where we put all the bad egg IP addresses
blacklist = set()
MAXVISITS = 15

ipwatcher = redis.Redis(db=5)

while True:
    _, addr = ipwatcher.blpop("ips")
    addr = ipaddress.ip_address(addr.decode("utf-8"))
    now = datetime.datetime.utcnow()
    addrts = f"{addr}:{now.minute}"
    n = ipwatcher.incrby(addrts, 1)
    if n >= MAXVISITS:
        print(f"Hat bot detected!:  {addr}")
        blacklist.add(addr)
    else:
        print(f"{now}:  saw {addr}")
    _ = ipwatcher.expire(addrts, 60)
```

در یک حلقه بی‌نهایت ابتدا از متد blpop استفاده میکنیم تا به صورت بلاکینگ از لیست ips داده بخوانیم. به ازای هر آیپی یک کلید به صورت ip:time میگذاریم که جای time زمان فعلی قرار میگیرد. با متد incby مقدار آن یکی زیاد میشود. زمان انقضای این کلید را شصت ثانیه میگذاریم زیرا میخواهیم در بازه شصت ثانیه‌ای ریکوئست زیاد دریافت نشود. در نهایت چک میکنیم تعداد بازدید در دقیقه از مقدار مجاز بیشتر شده یا خیر. حال کد زیر را اجرا میکنیم تا صحت تابع را چک کنیم. در این تابع ۲۰ ریکوئست پشت سر هم زده می‌شود:

```
>>> for _ in range(20):
...     r.lpush("ips", "104.174.118.18")
...
```

خروجی سمت کد به شکل زیر است:

```
→  DB python3 3.py
2021-12-24 20:15:20.167151:  saw 51.218.112.236
2021-12-24 20:15:20.168795:  saw 115.215.230.176
2021-12-24 20:15:20.170215:  saw 90.213.45.98
2021-12-24 20:15:20.171412:  saw 51.218.112.236
2021-12-24 20:17:34.015215:  saw 104.174.118.18
2021-12-24 20:17:34.016722:  saw 104.174.118.18
2021-12-24 20:17:34.018224:  saw 104.174.118.18
2021-12-24 20:17:34.019607:  saw 104.174.118.18
2021-12-24 20:17:34.021819:  saw 104.174.118.18
2021-12-24 20:17:34.032016:  saw 104.174.118.18
2021-12-24 20:17:34.033338:  saw 104.174.118.18
2021-12-24 20:17:34.034585:  saw 104.174.118.18
2021-12-24 20:17:34.037423:  saw 104.174.118.18
2021-12-24 20:17:34.039011:  saw 104.174.118.18
2021-12-24 20:17:34.040522:  saw 104.174.118.18
2021-12-24 20:17:34.041727:  saw 104.174.118.18
2021-12-24 20:17:34.042746:  saw 104.174.118.18
2021-12-24 20:17:34.043763:  saw 104.174.118.18
Hat bot detected!:  104.174.118.18
Hat bot detected!:  104.174.118.18
Hat bot detected!:  104.174.118.18
Hat bot detected!:  104.174.118.18
Hat bot detected!:  104.174.118.18
Hat bot detected!:  104.174.118.18
```

حال به ذخیره داده‌های پیچیده‌تر میپردازیم. فرض کنید یک داده به شکل زیر داریم:

```
>>> restaurant_484272 = {
...     "name": "Ravagh",
...     "type": "Persian",
...     "address": {
...         "street": {
...             "line1": "11 E 30th St",
...             "line2": "APT 1",
...         },
...         "city": "New York",
...         "state": "NY",
...         "zip": 10016,
...     }
... }
>>>
```

حال اگر بخواهیم با hmset آن را ذخیره کنیم به ارور زیر میخوریم:

```
>>> r.hmset(484272, restaurant_484272)
<stdin>:1: DeprecationWarning: Redis.hmset() is deprecated. Use Redis.hset() instead.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/sajializ/.local/lib/python3.8/site-packages/redis/commands/core.py", line
    return self.execute_command('HMSET', name, *items)
  File "/home/sajializ/.local/lib/python3.8/site-packages/redis/client.py", line 1071,
```

زیرا ردیس از داده‌های تو در تو یا nested پشتیبانی نمی‌کند. برای حل این مشکل چندین راه وجود دارد. یکی از راه‌ها تبدیل کردن داده به استرینگ است. مثلا:

```
>>> import json
>>> r.set(484272, json.dumps(restaurant_484272))
True
```

همانطورکه مشاهده میشود بدون هیچ مشکلی این داده را ذخیره کردیم. برای بازیابی آن حتما باید deserialize کنیم. یعنی:

```
>>> from pprint import pprint
>>> pprint(json.loads(r.get(484272)))
{'address': {'city': 'New York',
             'state': 'NY',
             'street': {'line1': '11 E 30th St', 'line2': 'APT 1'},
             'zip': 10016},
 'name': 'Ravagh',
 'type': 'Persian'}
>>>
```

همانطورکه میبینید با استفاده از json.loads توانستیم داده را بازیابی کنیم. با استفاده از کتابخانه yaml نیز این کار امکان‌پذیر بود.

راه دیگر استفاده از Delimiter است. در اینجا ما فرم ذخیره کردن داده را عوض میکنیم و از حالت nested خارج می‌کنیم. با استفاده از تابع زیر این کار را انجام می‌دهیم (فایل 4.py):

```python
def setflat_skeys(
    r: redis.Redis,
    obj: dict,
    prefix: str,
    delim: str = ":",
    *,
    _autopfix=""
) -> None:
    """Flatten `obj` and set resulting field-value pairs into `r`.

    Calls `.set()` to write to Redis instance inplace and returns None.

    `prefix` is an optional str that prefixes all keys.
    `delim` is the delimiter that separates the joined, flattened keys.
    `_autopfix` is used in recursive calls to created de-nested keys.

    The deepest-nested keys must be str, bytes, float, or int.
    Otherwise a TypeError is raised.
    """
    allowed_vtypes = (str, bytes, float, int)
    for key, value in obj.items():
        key = _autopfix + key
        if isinstance(value, allowed_vtypes):
            r.set(f"{prefix}{delim}{key}", value)
        elif isinstance(value, MutableMapping):
            setflat_skeys(
                r, value, prefix, delim, _autopfix=f"{key}{delim}"
            )
        else:
            raise TypeError(f"Unsupported value type: {type(value)}")
```

برای تست این تابع نیز کد زیر در ادامه آمده است:

```
restaurant_484272 = {
    "name": "Ravagh",
    "type": "Persian",
    "address": {
        "street": {
            "line1": "11 E 30th St",
            "line2": "APT 1",
        },
        "city": "New York",
        "state": "NY",
        "zip": 10016,
    }
}
r = redis.Redis()
r.flushdb()
setflat_skeys(r, restaurant_484272, 484272)
for key in sorted(r.keys("484272*")):
    print(f"{repr(key):35}{repr(r.get(key)):15}")
print(r.get("484272:address:street:line1"))
```

در ابتدا دیتابیس مربوطه را فلاش میکنیم تا داده‌های آن پاک شود. سپس از تابع تعریف شده استفاده می‌کنیم تا داده‌ها را از حالت nested در بیاورد و در ردیس ذخیره کند. داده جدید را برای نمایش پرینت میکنیم و در نهایت یکی از فیلدهای آن را نیز پرینت میکنیم. همانطورکه مشاهده میشود تودرتو بودن آن با استفاده از : نمایش داده شده است. نتیجه به شکل زیر است:

```
→  DB python3 4.py
b'484272:address:city'              b'New York'
b'484272:address:state'             b'NY'
b'484272:address:street:line1'      b'11 E 30th St'
b'484272:address:street:line2'      b'APT 1'
b'484272:address:zip'               b'10016'
b'484272:name'                      b'Ravagh'
b'484272:type'                      b'Persian'
b'11 E 30th St'
```

همانطورکه مشاهده میشود هر قسمت جدا شده است و با کلید مجزا ذخیره شده است و در نهایت نیز خط اول نام خیابان آدرس رستوران مورد نظر به درستی پرینت شده است.

مشکلات و توضیحات تکمیلی

خالی

آنچه آموختم / پیشنهادات

بسیار عالی بود. خلاصه و مفید.