自v2.0版本以后，CockroachDB允许通过第三方API存储和查询非结构化的[JSONB](https://www.cockroachlabs.com/docs/stable/jsonb.html)数据，同时数据库允许对该结构建立[倒排索引](https://www.cockroachlabs.com/docs/stable/inverted-indexes.html)来优化查询。

Step 1: 配置环境

选择一种运行环境，下载、安装并进行配置。

Golang

* 安装[CockroachDB](http://doc.cockroachchina.baidu.com/#quick-start/install-cockorachdb/)
* 安装[Go](https://golang.org/dl/)运行环境，版本不低于1.9

brew install go

* 安装[PostgreSQL驱动](https://github.com/lib/pq)

go get github.com/lib/pq

Python

* 安装[CockroachDB](http://doc.cockroachchina.baidu.com/#quick-start/install-cockorachdb/)
* 安装[Python psycopg2 驱动](http://initd.org/psycopg/docs/install.html)

pip install psycopg2

* 安装[Python Requests 库](http://docs.python-requests.org/en/master/)

pip install requests

Step 2: 启动1个节点

cockroach start --insecure --store=json-test --host=localhost

Step 3: 创建用户

执行[cockroach user](http://doc.cockroachchina.baidu.com/#deploy/access-management/manage-users/)命令创建用户：

cockroach user set maxroach --insecure

Step 4: 创建数据库和授权

* 启动交互式shell：

cockroach sql --insecure

* 执行SQL语句授权：

**CREATE** **DATABASE** jsonb\_test;

**SET** **DATABASE** = jsonb\_test;

**GRANT** **ALL** **ON** **DATABASE** jsonb\_test **TO** maxroach;

Step 5: 创建表

在SQL客户端当中执行：

**CREATE** **TABLE** programming (

id **UUID** **DEFAULT** uuid\_v4()::**UUID** **PRIMARY** **KEY**,

posts JSONB

);

**SHOW** **CREATE** **TABLE** programming;

+*--------------+-------------------------------------------------+*

| Table | CreateTable |

+*--------------+-------------------------------------------------+*

| programming | **CREATE** **TABLE** programming ( |

| | id **UUID** **NOT** NULL **DEFAULT** uuid\_v4()::**UUID**, |

| | posts JSON NULL, |

| | **CONSTRAINT** "primary" **PRIMARY** **KEY** (id **ASC**), |

| | FAMILY "primary" (id, posts) |

| | ) |

+*--------------+-------------------------------------------------+*

Step 6: 编写执行代码

Golang

* 此代码利用[Reddit API](https://www.reddit.com/dev/api/)查询[/r/programming](https://www.reddit.com/r/programming/)地址下的帖子，调用该API将返回一页内容，包含25个帖子。同时，用户可以通过"after"字段获取下一页的地址。
* 代码实现如下逻辑：
* 通过API发送请求
* 获取帖子内容并插入到数据库表，获取"after"字符串
* 根据"after"字符串获取下一次请求的地址
* 下载[json-sample.go](https://raw.githubusercontent.com/cockroachdb/docs/master/_includes/v2.0/json/json-sample.go)，代码实现了上述逻辑，内容如下：

**package** main

**import** (

"database/sql"

"fmt"

"io/ioutil"

"net/http"

"time"

\_ "github.com/lib/pq"

)

**func** main() {

db, err := sql.Open("postgres", "user=maxroach dbname=jsonb\_test sslmode=disable port=26257")

**if** err != nil {

panic(err)

}

*// The Reddit API wants us to tell it where to start from. The first request*

*// we just say "null" to say "from the start", subsequent requests will use*

*// the value received from the last call.*

after := "null"

**for** i := 0; i < 300; i++ {

after, err = makeReq(db, after)

**if** err != nil {

panic(err)

}

*// Reddit limits to 30 requests per minute, so don't do any more than that.*

time.Sleep(2 \* time.Second)

}

}

**func** makeReq(db \*sql.DB, after string) (string, error) {

*// First, make a request to reddit using the appropriate "after" string.*

client := &http.Client{}

req, err := http.NewRequest("GET", fmt.Sprintf("https://www.reddit.com/r/programming.json?after=%s", after), nil)

req.Header.Add("User-Agent", `Go`)

resp, err := client.Do(req)

**if** err != nil {

**return** "", err

}

res, err := ioutil.ReadAll(resp.Body)

**if** err != nil {

**return** "", err

}

*// We've gotten back our JSON from reddit, we can use a couple SQL tricks to*

*// accomplish multiple things at once.*

*// The JSON reddit returns looks like this:*

*// {*

*// "data": {*

*// "children": [ ... ]*

*// },*

*// "after": ...*

*// }*

*// We structure our query so that we extract the `children` field, and then*

*// expand that and insert each individual element into the database as a*

*// separate row. We then return the "after" field so we know how to make the*

*// next request.*

r, err := db.Query(`

INSERT INTO jsonb\_test.programming (posts)

SELECT json\_array\_elements($1->'data'->'children')

RETURNING $1->'data'->'after'`,

string(res))

**if** err != nil {

**return** "", err

}

*// Since we did a RETURNING, we need to grab the result of our query.*

r.Next()

**var** newAfter string

r.Scan(&newAfter)

**return** newAfter, nil

}

* 执行程序

**go** run json-sample.**go**

Python

* 此代码利用[Reddit API](https://www.reddit.com/dev/api/)查询[/r/programming](https://www.reddit.com/r/programming/)地址下的帖子，调用该API将返回一页内容，包含25个结果。同时，用户可以通过"after"字段获取下一页的地址。
* 代码实现如下逻辑：
* 通过API发送请求
* 获取"after"字符串
* 获取帖子内容并插入到数据库表
* 根据"after"字符串获取下一次请求的地址
* 下载[json-sample.py](https://raw.githubusercontent.com/cockroachdb/docs/master/_includes/v2.0/json/json-sample.py)，代码实现了上述逻辑，内容如下：

**import** json

**import** psycopg2

**import** requests

**import** time

conn = psycopg2.connect(database="jsonb\_test", user="maxroach", host="localhost", port=26257)

conn.set\_session(autocommit=**True**)

cur = conn.cursor()

*# The Reddit API wants us to tell it where to start from. The first request*

*# we just say "null" to say "from the start"; subsequent requests will use*

*# the value received from the last call.*

url = "https://www.reddit.com/r/programming.json"

after = {"after": "null"}

**for** n **in** range(300):

*# First, make a request to reddit using the appropriate "after" string.*

req = requests.get(url, params=after, headers={"User-Agent": "Python"})

*# Decode the JSON and set "after" for the next request.*

resp = req.json()

after = {"after": str(resp['data']['after'])}

*# Convert the JSON to a string to send to the database.*

data = json.dumps(resp)

*# The JSON reddit returns looks like this:*

*# {*

*# "data": {*

*# "children": [ ... ]*

*# },*

*# "after": ...*

*# }*

*# We structure our query so that we extract the `children` field, and then*

*# expand that and insert each individual element into the database as a*

*# separate row.*

cur.execute("""INSERT INTO jsonb\_test.programming (posts)

SELECT json\_array\_elements(%s->'data'->'children')""", (data,))

*# Reddit limits to 30 requests per minute, so don't do any more than that.*

time.sleep(2)

cur.close()

conn.close()

* 执行程序

python json-sample.py

Step 7: 查询数据

启动交互式shell，执行SQL查询：

**SELECT** **count**(\*) **FROM** programming;

+*-------+*

| count |

+*-------+*

| 1120 |

+*-------+*

**SELECT** **count**(\*) **FROM** programming;

+*-------+*

| count |

+*-------+*

| 2400 |

+*-------+*

// 检索所有指向github域名地址的对象id

**SELECT** id **FROM** programming **WHERE** posts @> '{"data": {"domain": "github.com"}}';

+*--------------------------------------+*

| id |

+*--------------------------------------+*

| 0036d489-3fe3-46ec-8219-2eaee151af4b |

| 00538c2f-592f-436a-866f-d69b58e842b6 |

| 00aff68c-3867-4dfe-82b3-2a27262d5059 |

| 00cc3d4d-a8dd-4c9a-a732-00ed40e542b0 |

| 00ecd1dd-4d22-4af6-ac1c-1f07f3eba42b |

| 012de443-c7bf-461a-b563-925d34d1f996 |

| 014c0ac8-4b4e-4283-9722-1dd6c780f7a6 |

| 017bfb8b-008e-4df2-90e4-61573e3a3f62 |

| 0271741e-3f2a-4311-b57f-a75e5cc49b61 |

| 02f31c61-66a7-41ba-854e-1ece0736f06b |

| 035f31a1-b695-46be-8b22-469e8e755a50 |

| 03bd9793-7b1b-4f55-8cdd-99d18d6cb3ea |

| 03e0b1b4-42c3-4121-bda9-65bcb22dcf72 |

| 0453bc77-4349-4136-9b02-3a6353ea155e |

...

+*--------------------------------------+*

(334 rows)

Time: 105.877736ms

**NOTE:** 因为数据源的因素，读者操作时查询结果内容可能跟上述有所区别。

Step 8: 建立倒排索引优化性能

执行SQL语句，在posts字段上建立倒排索引：

**CREATE** INVERTED **INDEX** **ON** programming(posts);

Step 9: 再次查询

执行与**Step 7**相同的SQL语句：

**SELECT** id **FROM** programming **WHERE** posts @> '{"data": {"domain": "github.com"}}';

...

(334 rows)

Time: 28.646769ms

**Step 7**的查询花费了105.877736ms，而建立索引后查询时间缩短到28.646769ms。