TESTE DE PERFORMANCE DO BANCO DE DADOS CASSANDRA

João Pedro Figueirôa Nascimento,
Brauliro Gonçalves Leal.

RESUMO: Foi feita uma avaliação de performance do gerenciador de banco de dados Apache Cassandra 3.11.3 ao executar 60 blocos de 500 mil operações (insert, select e update). A duração média das operações unitárias (DOU) de insert, select e update foram iguais a 0,000566, 0,001015 e 0,000569 s, respectivamente. As equações ajustadas para os valores de duração acumulada das operações (DAO) em função do número de operações (NOP) foram DAO = 0,000566297513789 × NOP + 22,7190724266566, com R^2 = 0,999987226327609 para as operações de insert; DAO = 0,001014740644349 × NOP + 7,23468197334842, com R^2 = 0,999997010621452, para as operações de select; e DAO = 0,000569127180016 × NOP + 41.7702037670588, com R^2 = 0.999978549020038, para as operações de update. As operações de update e insert requereram, respectivamente, 56,07 e 55,82% do tempo de select. Embora todas as operações realizem acesso ao banco de dados, select utilizam mais tempo computacional. Os resultados indicaram que o banco de dados Cassandra mantém sua performance mesmo em frente a uma grande quantidade de registros.
PALAVRAS-CHAVE: Banco de Dados, NoSQL, Performance

PERFORMANCE TEST OF CASSANDRA DATABASE

ABSTRACT: A Apache Cassandra 3.11.3 database was used to insert 60 blocks of 500 thousand data to verify the database performance in situations of insertion, selection and update. The average duration from the single operation insert, select and update were 0,000566, 0,001015 e 0,000569 s respectively. The adjusted equations to the values of operations accumulated duration (DAO) in function of the number of operations (NOP) were DAO = 0,000566297513789 × NOP + 22,7190724266566, with R^2 = 0,999987226327609 to insert operations; DAO = 0,001014740644349 × NOP + 7,23468197334842, with R^2 = 0,999997010621452, to select operations; and DAO = 0,000569127180016 × NOP + 41.7702037670588, with R^2 = 0.999978549020038, to update operations. Update and insert operations require 56,07 and 55,82% of select time operation respectively. Although all operations have access to database, select uses more computational time.
KEYWORDS: Database, NoSQL, Performance

INTRODUÇÃO:

Bancos de dados são estruturas que armazenam dados. São úteis para manipulação de registros, permitindo acessar, modificar, incluir e remover dados. Os bancos de dados, em geral, são gerenciados por Sistemas de Gerenciamento de Banco de Dados (SGBD) cujo principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, manipulação e organização dos dados.

Apache Cassandra é um SGBD NoSQL (Not Only SQL - Não Apenas SQL, em tradução livre), tipos de banco de dados não relacionais, do tipo coluna. O Cassandra promete uma alta escalabilidade e disponibilidade sem a perda de performance, significando que a quantidade de nós utilizados para o armazenamento de dados pode crescer sem muitas dificuldades e pode ser utilizado por aplicações onde a perda de dados é inconcebível. Exemplos relevantes do emprego de Cassandra são GitHub e Netflix.

O presente trabalho visa testar através de um programa sintético escrito em Python 3.7, com o auxílio da aplicação web Jupyter Notebook a performance do banco de dados Apache Cassandra em função de múltiplas inserções, seleções e atualizações de dados por um determinado período de tempo.

MATERIAIS E MÉTODOS:

Para utilização do banco de dados Apache Cassandra se faz necessária a instalação do mesmo, no caso a versão 3.11.3, tal instalação pode ser concluída através da leitura do website <http://cassandra.apache.org/download/>.

Foi criada uma tabela com oito campos conforme descrito no Quadro 01.

table	
A0	UUID
A1	INT
A2	BIGINT
A3	DOUBLE
A4	VARCHAR(26)
A5	TEXT
A6	DATE
A7	TIMESTAMP

Quadro 01 - Tabela (table) inserida no banco de dados.

Foi desenvolvido um programa em Python 3.7 para realizar 30 milhões de operações insert, select e update. O desempenho foi analisado através da duração acumulada das operações e da duração média da operações unitárias por bloco de insert, select e update em função do número de operações.

Utilizou-se o driver para Cassandra para Python 3.7 para conexão com banco de dados e a biblioteca *time* do Python para marcação do tempo. Foi utilizada uma máquina com as seguintes configurações: CPU Core i7, 3,40 GHz e 16 GB de RAM rodando sobre o sistema operacional Ubuntu 16.04.

O driver pode ser instalado utilizando o comando a seguir e para mais detalhes atentar ao website <https://datastax.github.io/python-driver/installation.html>.

```
pip install cassandra-driver
```

RESULTADO E DISCUSSÃO:

Quadro 02 apresenta a duração acumulada das operações (DAO) e a duração média das operações unitárias (DOU) por bloco de insert, select e update em função do número de operações.

Os valores da DAO variaram de 291 a 16692,35 s; 487 a 30439,12 s; e 287 a 17067,89 s para as operações de insert, select e update, respectivamente. Por outro lado, os valores médios da DOU para as operações insert, select e update foram iguais 0,000566; 0,001015; e 0,000569 s, respectivamente. As figuras 1.a, 1.c e 1.e mostram os gráficos da variação dos valores do DAO em função do número de operações insert, select e update, respectivamente. As figuras 1.b, 1.d, 1.f mostram os gráficos da variação dos valores do DOU em função do número de operações insert, select e update, respectivamente. As figuras 1.a, 1.c e 1.e também apresentam equações ajustadas para os valores do DAO em função do número de operações; sendo DAO = 0,000566297513789 × NOP + 22,7190724266566, com R^2 = 0,999987226327609 para as operações de insert; DAO = 0,001014740644349 × NOP + 7,23468197334842, com R^2 = 0,999997010621452, para as operações de select; e DAO = 0,000569127180016 × NOP + 41.7702037670588, com R^2 = 0.999978549020038, para as operações de update.

NOP (x500000)	**Insert**		**Select**		**Update**	
	DAO (s)	**DOU (s)**	**DAO (s)**	**DOU (s)**	**DAO (s)**	**DOU (s)**
1	291.447015	0.00058289403	487.788809	0.000975577618	287.538619	0.000575077238
2	573.354952	0.000573354952	993.310457	0.000993310457	582.381148	0.000582381148
3	858.310141	0.000572206760667	1500.68541	0.00100045694	866.415526	0.000577610350667
4	1142.939507	0.0005714697535	2014.218126	0.001007109063	1161.574007	0.000580787035
5	1430.415605	0.000572166242	2509.69787	0.001003879148	1453.217509	0.000581287036
6	1705.975641	0.000568658547	3028.616355	0.001009538785	1730.07418	0.000576691393333
...	...	...	...	...	...	...
57	16126.290562	0.000565834756561	28917.054726	0.001014633499158	16251.543226	0.000570229586877
58	16419.790168	0.000566199660966	29419.029302	0.001014449286276	16539.692287	0.000570334216793
59	16704.082136	0.000566240072407	29928.843923	0.001014537082136	16781.570009	0.000568866779966
60	16992.348836	0.000566411627867	30439.118664	0.001014637288	17067.891498	0.000568929166

Média	-	0.000568760382817	-	0.0010140221 4967	-	0.0005734907 4684

Quadro 02 - Duração acumulada das operações (DAO) e a duração média das operações unitárias (DOU) por bloco de insert, select e update e função do número de operações (NOP) (Amostragem).

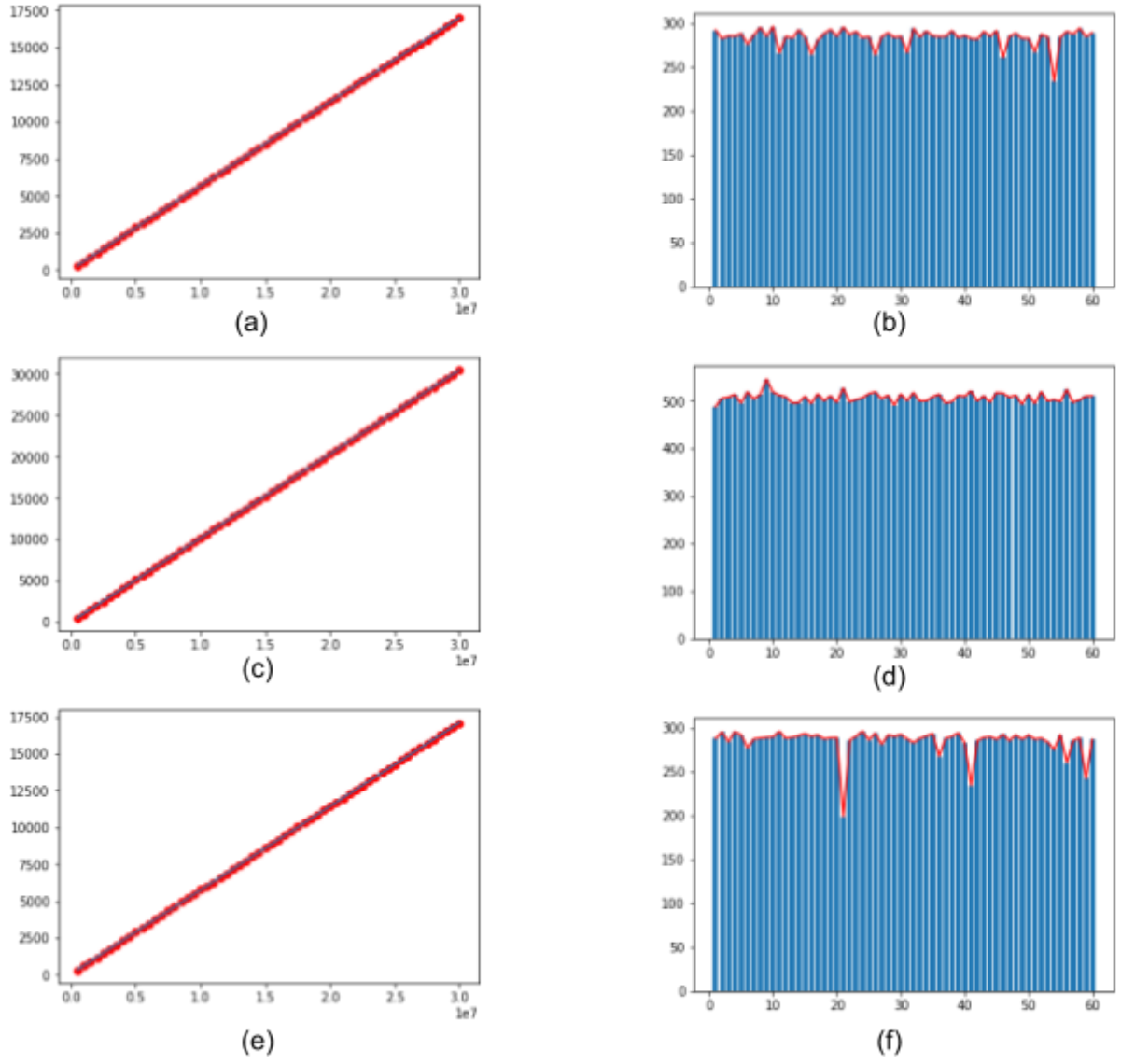

Figura 01 - Resultados.

Das três operações avaliadas a que apresentou maior tempo de execução foram as de insert, seguida pelas de update que, por sua vez exigiram mais tempo que as de select. As operações de update e insert requereram, respectivamente, 56,07 e 55,82% do tempo que as de select. Embora todas as operações realizem acesso ao banco de dados, select utilizam mais tempo computacional.

Conforme pode-se observar nas Figuras 1.a, 1.c e 1.e, os valores de DAO variaram de modo linear para as três operações.

CONCLUSÃO:

Avaliou-se a performance do Apache Cassandra ao executar 30 milhões de operações (insert, select, update), cuja durações médias foram de 0,57, 1,01 e 0,57 ms para insert, select e update respectivamente. Operações de update e insert requerem 56,07 e 55,82% do tempo de select, respectivamente.

As equações ajustadas para os valores duração acumulada das operações (DAO) em função do número de operações (NOP) foram DAO = 0,000566297513789 × NOP + 22,7190724266566, com R^2 = 0,999987226327609 para as operações de insert; DAO = 0,001014740644349 × NOP + 7,23468197334842, com R^2 = 0,999997010621452, para as operações de select; e DAO = 0,000569127180016 × NOP + 41.7702037670588, com R^2 = 0.999978549020038, para as operações de update.

Pode-se observar que o banco de dados Cassandra é de fato um banco de dados que mantém sua eficiência com grandes quantidades de dados, como possível notar pela figura 1 b, d e f, que mantém o comportamento constante independente da quantidade de registros presentes na tabela.

BIBLIOGRAFIA:

[1] Apache Cassandra. Disponível em http://cassandra.apache.org/
[2] NoSQ. Disponível em https://pt.wikipedia.org/wiki/NoSQL
[3] Python Cassandra Driver. Disponível em https://datastax.github.io/python-driver/

ANEXO 1:

Tabela de resultados completa:

NOP (x500000)	Insert		Select		Update	
	DAO (s)	DOU (s)	DAO (s)	DOU (s)	DAO (s)	DOU (s)
1	291.447015	0.00058289403	487.788809	0.000975577618	287.538619	0.000575077238
2	573.354952	0.000573354952	993.310457	0.000993310457	582.381148	0.000582381148
3	858.310141	0.000572206760667	1500.68541	0.0010004569 4	866.415526	0.0005776103 50667
4	1142.939507	0.000571469753 5	2014.218126	0.0010071090 63	1161.574007	0.0005807870 035
5	1430.415605	0.000572166242	2509.69787	0.0010038791 48	1453.217509	0.0005812870 036
6	1705.975641	0.000568658547	3028.616355	0.0010095387 85	1730.07418	0.0005766913 93333
7	1992.012753	0.000569146500 857	3532.500676	0.0010092859 07429	2017.051973	0.0005763005 63714
8	2286.38614	0.000571596535	4045.539426	0.0010113848 565	2304.870124	0.0005762175 31
9	2570.939532	0.000571319896	4591.656219	0.0010203680 48667	2593.765679	0.0005763923 73111
10	2866.402875	0.000573280575	5109.905906	0.0010219811 812	2883.204564	0.0005766409 128
11	3131.987492	0.000569452271 273	5621.899063	0.0010221634 66	3178.741675	0.0005779530 31818
12	3416.303788	0.000569383964 667	6130.311665	0.0010217186 10833	3466.418909	0.0005777364 84833
13	3698.675252	0.000569026961 846	6626.503303	0.0010194620 46615	3755.333696	0.0005777436 45538
14	3990.239403	0.000570034200 429	7122.68969	0.0010175270 98571	4046.055784	0.0005780079 69143

NOP (x500000)	Insert		Select		Update	
	DAO (s)	DOU (s)	DAO (s)	DOU (s)	DAO (s)	DOU (s)
16	4538.198145	0.000567274768125	8126.928869	0.001015866108625	4628.85223	0.000578606528875
17	4818.286577	0.000566857244353	8641.608852	0.001016659864941	4920.418901	0.000578872811882
18	5105.953762	0.000567328195778	9141.915747	0.001015768416333	5207.432357	0.000578603595222
19	5398.132299	0.000568224452526	9652.622823	0.001016065560316	5495.551247	0.000578479078632
20	5682.832238	0.0005682832238	10149.631239	0.001014963239	5784.029308	0.000578402308
21	5977.545187	0.000569290017781	10676.456578	0.001016805388381	5982.7973	0.000569790219048
22	6263.784419	0.000569434947182	11174.217053	0.001015837913909	6268.294705	0.000569844973182
23	6553.65729	0.000569883242609	11677.188391	0.001015407686174	6558.471659	0.000570301883391
24	6836.304529	0.000569692044083	12183.488584	0.001015290715333	6854.256829	0.000571188069083
25	7120.453321	0.00056963626568	12698.216274	0.001015857300192	7140.256541	0.000571220502328
26	7384.022664	0.000568001743385	13217.040695	0.001016695438077	7433.669216	0.000571820708923
27	7668.179931	0.000568013328222	13721.467723	0.001016405016519	7714.623446	0.000571453588593
28	7956.446126	0.000568317580429	14233.477652	0.001016676975143	8006.115362	0.00057186538383
29	8239.624315	0.000568249952759	14724.476919	0.001015481166828	8296.060005	0.000572142006931
30	8524.045241	0.000568269682733	15238.770056	0.001015918003733	8588.085185	0.000572539012333

NOP (x500000)	Insert		Select		Update	
	DAO (s)	DOU (s)	DAO (s)	DOU (s)	DAO (s)	DOU (s)
32	9083.225195	0.000567701574688	16256.716841	0.001016044802563	9157.965496	0.000572372843 5
33	9367.027614	0.000567698643273	16756.233533	0.001015529305 03	9445.524986	0.000572456059758
34	9657.213686	0.000568071393294	17256.197928	0.001015070466353	9735.921588	0.000572701269882
35	9942.270487	0.000568129742114	17764.626202	0.001015121497257	10028.784725	0.000573073412857
36	10226.392035	0.000568132890833	18278.806669	0.001015489259389	10296.235491	0.000572013082833
37	10510.819251	0.000568152391946	18774.585386	0.001014842453297	10583.806263	0.000572097635838
38	10801.470431	0.000568498443737	19272.776681	0.001014356667421	10873.752924	0.000572302785474
39	11084.463451	0.000568434023128	19784.041505	0.001014566231026	11167.463383	0.000572690429897
40	11370.144739	0.000568507236 95	20293.506872	0.001014675 3436	11450.296394	0.000572514 8197
41	11652.000246	0.000568390255902	20814.062033	0.001015320099171	11684.521095	0.000569976663878
42	11933.387591	0.000568256551952	21314.070555	0.001014955740714	11969.808286	0.000569990870762
43	12223.128268	0.000568517593 86	21824.379971	0.001015087440512	12258.409122	0.000570158563814
44	12507.830991	0.000568537772318	22321.807987	0.001014627635773	12547.935791	0.000570360717773
45	12798.449537	0.000568819979422	22839.232771	0.001015077012044	12834.488945	0.000570421730889
46	13058.68509	0.000567768916957	23355.131086	0.001015440482	13126.465995	0.000570715912826

NOP (x500000)	Insert		Select		Update	
	DAO (s)	DOU (s)	DAO (s)	DOU (s)	DAO (s)	DOU (s)
48	13630.731497	0.000567947145708	24374.597193	0.001015608216375	13703.074941	0.000570961455875
49	13912.884703	0.000567872845 02	24867.787533	0.001015011736041	13989.816385	0.000571012913673
50	14194.972476	0.000567798899 04	25381.478073	0.001015259122292	14281.309627	0.000571252385 08
51	14461.899531	0.000567133314941	25876.100471	0.001014749038078	14568.008383	0.000571294446392
52	14748.624705	0.000567254796346	26395.124545	0.001015197097885	14855.845138	0.000571378659154
53	15032.456104	0.000567262494491	26894.752592	0.001014896324226	15139.287048	0.000571293850868
54	15265.932145	0.000565404894259	27397.688453	0.001014729201963	15414.302058	0.000570900076222
55	15549.370458	0.000565431653018	27896.251853	0.001014409158291	15706.030713	0.000571128389564
56	15839.369752	0.000565691776857	28419.998417	0.001014999943464	15966.409564	0.000570228913
57	16126.290562	0.000565834756561	28917.054726	0.001014633499158	16251.543226	0.000570229586877
58	16419.790168	0.000566199660966	29419.029302	0.001014449286276	16539.692287	0.000570334216793
59	16704.082136	0.000566240072407	29928.843923	0.001014537082136	16781.570009	0.000568866779966
60	16992.348836	0.000566411627867	30439.118664	0.001014637 2888	17067.891498	0.000568929 7166
Média	-	0.000568760382817	-	0.001014022 14967	-	0.000573490 74684

ANEXO 2:

Resumo do código utilizado, o código completo é possível encontrar no endereço <https://github.com/joaopedrofn/CassandraAnalysis>, onde também estão disponiveis este trabalho e a tabela de resultados sem tratamento.

```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
import time

## Database configuration

cluster = Cluster(['localhost'],
auth_provider=PlainTextAuthProvider('cassandra', 'cassandra'))
session = cluster.connect()

# Create KeySpace
session.execute('CREATE KEYSPACE IF NOT EXISTS analysis WITH replication
= {\'class\': \'SimpleStrategy\', \'replication_factor\': 1}')

# Use KeySpace created
session.set_keyspace('analysis')

# Create Table
session.execute("CREATE TABLE IF NOT EXISTS \"table\" ("+
                "A0 uuid, "+
                "A1 int PRIMARY KEY, "+
                "A2 bigint, "+
                "A3 double, "+
                "A4 varchar, "+
                "A5 text, "+
                "A6 date, "+
                "A7 timestamp )")
rows = session.execute('select * from system_schema.columns')
print('+---------------------------------+')
print('|              table              |')
print('+---------------------------------+')
for row in rows:
    if row.table_name == 'table':
        print('|', row.column_name, '|', row.type, '
'*(9-len(row.type)), '|',  row.kind, ' '*(13-len(row.kind)), '|')
print('+---------------------------------+')

## Meta data
```

```
# * R is the number of iterations <br />
# * BI is the size of a block of insertions<br />
# * BU is the size of a block of updates<br />
# * BS is the size of a block of selections<br />

R = 60
BI = BU = BS = 500000
categories = [i+1 for i in range(R)]

## Definition of functions

def insert(control):
    initialTime = time.time()
    for i in range(BI):
        session.execute('INSERT INTO "table" (A0, A1, A2, A3, A4, A5,
A6, A7) VALUES ('+
                        'uuid(), '+
                        str(control)+', '+
                        '9223372036854775807, '+
                        '1.2, '+
                        '\'abcdefghijklmnopqrstuvwxy\', '+
                        '\'Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Integer ultricies lorem metus, vel finibus risus
convallis sit amet. Nam auctor ex et ipsum euismod, vel consectetur eros
blandit. Sed sit amet enim vitae nisi varius molestie. Maecenas in
tortor sem. Ut sit amet lobortis erat, ac egestas libero. Nunc id purus
sodales, dictum massa gravida, condimentum diam. Maecenas eu vulputate
nunc, vitae tempor odio. Orci varius natoque penatibus et magnis dis
parturient montes, nascetur ridiculus mus. Duis vitae turpis quam.
Maecenas at dui at justo vehicula scelerisque eu in eros. Sed volutpat,
magna nec pretium tincidunt, risus nibh posuere mauris, quis feugiat
augue lorem ut nunc. Lorem ipsum dolor sit amet, consectetur adipiscing
elit. In eu tellus nec nulla ultricies efficitur. Interdum et malesuada
fames ac ante ipsum primis in faucibus. \', '+
                        'toDate(dateof(now())), '+
                        'dateof(now())'+
                        ')')
        control += 1
    return (time.time() - initialTime), control

def update(control):
    initialTime = time.time()
    for i in range(BI):
        session.execute('UPDATE "table" SET '+
                        'A2 = 9223372036854775800, '+
```

```
                        'A3 = 1.3, '+
                        'A4 = \'abbdefghijklmnopqrstuvwxy\', '+
                        'A5 = \'Lorem ipsum dolor site amet, consectetur
adipiscing elit. Integer ultricies lorem metus, vel finibus risus
convallis sit amet. Nam auctor ex et ipsum euismod, vel consectetur eros
blandit. Sed sit amet enim vitae nisi varius molestie. Maecenas in
tortor sem. Ut sit amet lobortis erat, ac egestas libero. Nunc id purus
sodales, dictum massa gravida, condimentum diam. Maecenas eu vulputate
nunc, vitae tempor odio. Orci varius natoque penatibus et magnis dis
parturient montes, nascetur ridiculus mus. Duis vitae turpis quam.
Maecenas at dui at justo vehicula scelerisque eu in eros. Sed volutpat,
magna nec pretium tincidunt, risus nibh posuere mauris, quis feugiat
augue lorem ut nunc. Lorem ipsum dolor sit amet, consectetur adipiscing
elit. In eu tellus nec nulla ultricies efficitur. Interdum et malesuada
fames ac ante ipsum primis in faucibus. \', '+
                        'A6 = toDate(dateof(now())), '+
                        'A7 = dateof(now())'+
                        'WHERE A1 = '+str(control))
        control += 1
    return (time.time() - initialTime), control

def select(control):
    initialTime = time.time()
    for i in range(BI):
        session.execute('SELECT * FROM "table" WHERE A1 =
'+str(control))
        control += 1
    return (time.time() - initialTime), control

## Storage

insertTimes = []
updateTimes = []
selectTimes = []

## DO IT

control = 1
for j in range(R):
    control = BI*j + 1
    t, control = insert(control)
    print('Insert', control)
    insertTimes.append(t)
    control = BI*j + 1
    t, control = update(control)
```

```
    print('Update', control)
    updateTimes.append(t)
    control = BI*j + 1
    t, control = select(control)
    print('Select', control)
    selectTimes.append(t)
```