**Estudio comparativo de rendimiento en el acceso a bases de datos**  
Por: Tomás Guzmán (21615008) y Antonio Blanco (20613680)

# Actividad 1: elaboración de scripts para los escenarios 1, 3 y 5

## Para MySQL

from multiprocessing import Process

from time import time

from random import randint

import threading

import MySQLdb

import sys

import os

class MySQLTest:

def \_\_init\_\_(self, database):

self.connection = MySQLdb.connect(host="localhost",

user="root",

passwd="\*\*\* ",

db=database)

self.cursor = self.connection.cursor()

self.lock = threading.Lock()

def commit(self):

self.connection.commit()

def close(self):

self.connection.close()

def insert(self, min\_limit, max\_limit):

'''Insert a number of elements into the table table'''

for i in range(min\_limit, max\_limit):

cursor.execute('INSERT INTO test VALUES(%d, "%s", %d)' %

(i, "ESTE ES EL TEXTO DE PRUEBA NUMERO "+str(i), i))

self.connection.commit()

def select\_with\_index(self, index):

'''Select a random record using indexes'''

cursor.execute('SELECT \* FROM test WHERE id=%d' % index)

def select(self, pseudo\_index):

'''Select a random record using an not indexed id'''

cursor.execute('SELECT \* FROM test WHERE id2=%d' % pseudo\_index)

if \_\_name\_\_ == "\_\_main\_\_":

test = MySQLTest('mysqltest')

cursor = test.cursor

# Create test table

try:

cursor.execute('CREATE TABLE test('

'id INT PRIMARY KEY AUTO\_INCREMENT,'

'texto VARCHAR(255),'

'id2 INT)') # remember, 255 chars for MySQL

print('CREATE TABLE')

exit()

# Create index for the test table

cursor.execute('CREATE INDEX id ON test (id)')

print('CREATE INDEX')

# Fill table with 1000000 elements

print("Filling...")

test.insert(0, 1000000)

print("xd")

exit()

except MySQLdb.Error as e:

pass

print("Success!\n")

try:

elements = [10, 1000, 100000, 300000]

process = [1, 10, 50]

j = 1000000 # Hardcoded, yes!

# Escenario 1

print("\nScenario 1:")

for e in elements:

for proc in process:

start = time()

what = e/proc

print('ins = %d \t procs = %d \t per = %d'

% (e, proc, what))

for i in range(0, proc):

min\_limit = int(j + what \* i)

max\_limit = int(j + what \* (i + 1))

p = Process(target=test.insert,

args=(min\_limit, max\_limit))

p.start()

p.join()

end = time()

elapsed = end - start

j += e

print(j)

print("Time elapsed: %.5f\n" % elapsed)

# Escenario 3

print("\nScenario 3:")

for e in elements:

for proc in process:

start = time()

what = e/proc

print('ins = %d \t procs = %d \t per = %d'

% (e, proc, what))

for i in range(0, proc):

p = Process(target=test.select\_with\_index,

args=(randint(1, e),))

p.start()

p.join()

end = time()

elapsed = end - start

print("Time elapsed: %.5f\n" % elapsed)

# Escenario 5

print("\nScenario 5:")

for e in elements:

for proc in process:

start = time()

what = e/proc

print('ins = %d \t procs = %d \t per = %d'

% (e, proc, what))

for i in range(0, proc):

p = Process(target=test.select,

args=(randint(1, e),))

p.start()

p.join()

end = time()

elapsed = end - start

print("Time elapsed: %.5f\n" % elapsed)

except MySQLdb.Error as e:

print("DB error: {} s".format(e))

test.close()

## Para SQLite

from multiprocessing import Process

from time import time

from random import randint

import threading

import sqlite3

import sys

import os

class SQLiteTest:

def \_\_init\_\_(self, database):

self.connection = sqlite3.connect(database)

self.cursor = self.connection.cursor()

self.lock = threading.Lock()

def commit(self):

self.connection.commit()

def close(self):

self.connection.close()

def insert(self, min\_limit, max\_limit):

'''Insert a number of elements into the table table'''

for i in range(min\_limit, max\_limit):

cursor.execute('INSERT INTO test VALUES(?, ?, ?)',

(i, "ESTE ES EL TEXTO DE PRUEBA NUMERO "+str(i), i))

self.connection.commit()

def select\_with\_index(self, index):

'''Select a random record using indexes'''

cursor.execute('SELECT \* FROM test WHERE id=?', (index,))

def select(self, pseudo\_index):

'''Select a random record using an not indexed id'''

cursor.execute('SELECT \* FROM test WHERE id2=?', (pseudo\_index,))

if \_\_name\_\_ == "\_\_main\_\_":

test = SQLiteTest('sqlite-test.db')

cursor = test.cursor

# Create test table

try:

cursor.execute('CREATE TABLE test('

'id integer PRIMARY KEY,'

'texto text,'

'id2 integer)') # remember, 255 chars for MySQL

print('CREATE TABLE')

# Create index for the test table

cursor.execute('CREATE INDEX id ON test (id)')

print('CREATE INDEX')

# Fill table with 1000000 elements

print("Filling...")

test.insert(0, 1000000)

except sqlite3.Error as e:

pass

print("Success!\n")

try:

elements = [10, 1000, 100000, 300000]

process = [1, 10, 50]

j = 1000000 # Hardcoded, yes!

# Escenario 1

print("\nScenario 1:")

for e in elements:

for proc in process:

start = time()

what = e/proc

print('ins = %d \t procs = %d \t per = %d'

% (e, proc, what))

for i in range(0, proc):

min\_limit = int(j + what \* i)

max\_limit = int(j + what \* (i + 1))

p = Process(target=test.insert,

args=(min\_limit, max\_limit))

p.start()

p.join()

end = time()

elapsed = end - start

j += e

print(j)

print("Time elapsed: %.5f\n" % elapsed)

# Escenario 3

print("\nScenario 3:")

for e in elements:

for proc in process:

start = time()

what = e/proc

print('ins = %d \t procs = %d \t per = %d'

% (e, proc, what))

for i in range(0, proc):

p = Process(target=test.select\_with\_index,

args=(randint(1, e),))

p.start()

p.join()

end = time()

elapsed = end - start

print("Time elapsed: %.5f\n" % elapsed)

# Escenario 5

print("\nScenario 5:")

for e in elements:

for proc in process:

start = time()

what = e/proc

print('ins = %d \t procs = %d \t per = %d'

% (e, proc, what))

for i in range(0, proc):

p = Process(target=test.select,

args=(randint(1, e),))

p.start()

p.join()

end = time()

elapsed = end - start

print("Time elapsed: %.5f\n" % elapsed)

except sqlite3.Error as e:

print("DB error: {} s".format(e))

test.close()

# Actividad 2: llenar las tablas para los escenarios indicados

## MySQL

### Escenario 1: INSERT mono y multi-proc

|  |  |  |  |
| --- | --- | --- | --- |
| #procesos vs INSERTS adicionales | 1 | 10 | 50 |
| 10 | 0.01082 | 0.08417 | 0.15093 |
| 1.000 | 0.10563 | 0.19695 | 0.56664 |
| 1000.000 | 6.75609 | 6.53192 | 7.302 |
| 300.000 | 20.69562 | 19.726 | 19.74022 |

### Escenario 3: SELECT mono y multi proceso con índice

|  |  |  |  |
| --- | --- | --- | --- |
| #procesos concurrentes vs #SELECT | 1 | 10 | 50 |
| 10 | 0.00355 | 0.03591 | 0.18512 |
| 1.000 | 0.00340 | 0.03539 | 0.18159 |
| 1000.000 | 0.00359 | 0.03570 | 0.18427 |
| 300.000 | 0.00368 | 0.03899 | 0.18242 |

### Escenario 5: SELECT mono y multi proceso sin índice

|  |  |  |  |
| --- | --- | --- | --- |
| #procesos concurrentes / #SELECT | 1 | 10 | 50 |
| 10 | 0.27323 | 3.01464 | 13.64915 |
| 1.000 | 0.27031 | 2.70063 | 13.72949 |
| 1000.000 | 0.26930 | 2.70089 | 13.39918 |
| 300.000 | 0.27022 | 2.68862 | 13.47036 |

## SQLite

### Escenario 1: INSERT mono y multi-proc

|  |  |  |  |
| --- | --- | --- | --- |
| #procesos concurrentes /  #INSERTS adicionales | 1 | 10 | 50 |
| 10 | 0.06178 | 0.23067 | 0.38499 |
| 1.000 | 0.02443 | 0.31018 | 1.47389 |
| 1000.000 | 0.67207 | 1.02054 | 2.67676 |
| 300.000 | 2.06394 | 2.23296 | 3.43475 |

### Escenario 3: SELECT mono y multi proceso con índice

|  |  |  |  |
| --- | --- | --- | --- |
| #procesos concurrentes / #SELECT | 1 | 10 | 50 |
| 10 | 0.00496 | 0.05931 | 0.24655 |
| 1.000 | 0.00467 | 0.05244 | 0.24230 |
| 1000.000 | 0.00457 | 0.05527 | 0.24284 |
| 300.000 | 0.00479 | 0.05153 | 0.24108 |

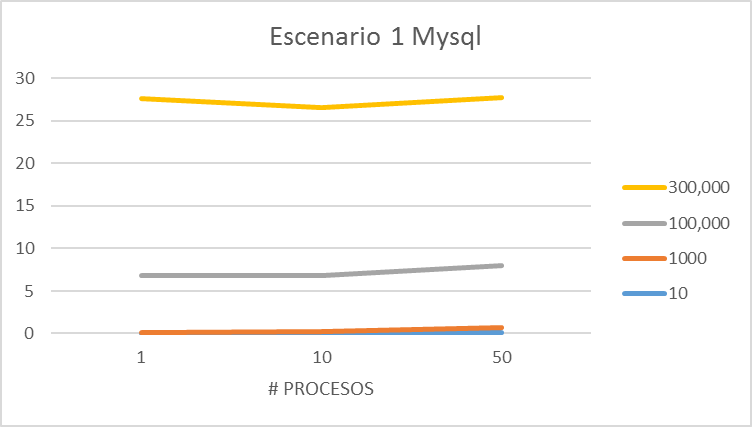
### Escenario 5: SELECT mono y multi proceso sin índice

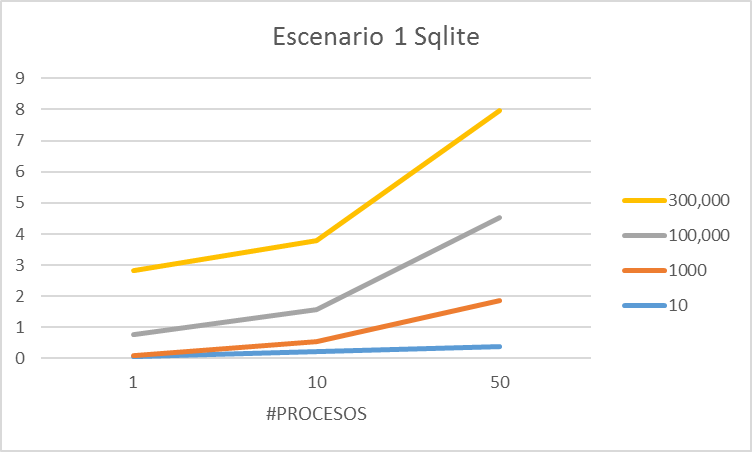
|  |  |  |  |
| --- | --- | --- | --- |
| #procesos concurrentes / #SELECT | 1 | 10 | 50 |
| 10 | 0.00476 | 0.05219 | 0.23817 |
| 1.000 | 0.00464 | 0.05190 | 0.24812 |
| 1000.000 | 0.00473 | 0.09339 | 0.54723 |
| 300.000 | 0.00762 | 0.24693 | 0.98402 |

# Actividad 3: Análisis de rendimiento

## Escenario 1: INSERT mono y multiproceso

Ambos manejadores de bases de datos sufren en su rendimiento con la concurrencia y con el volumen de entradas (INSERTS). Sin embargo, se puede observar que MySQL sufre mucho más que SQLite (un resultado poco esperado, pues se supone que SQLite no está de acuerdo con la concurrencia de usuarios). Los tiempos de MySQL son mucho mayores, aun cuando hay solo un proceso insertando récords en la misma. A nivel de sistema operativo, los procesos son repartidos casi en partes iguales entre la cantidad de procesadores físicos que la computadora posea, de tal forma, no se ve mucho incremento en el rendimiento al entregar cincuenta procesos al CPU, dando casi diez procesos por procesador físico. Se esperaría un mejor rendimiento con menos procesos contiguos. Véase que en ambos manejadores, diez procesos concurrentes se ejecutan en menor tiempo que los cincuenta procesos concurrentes.

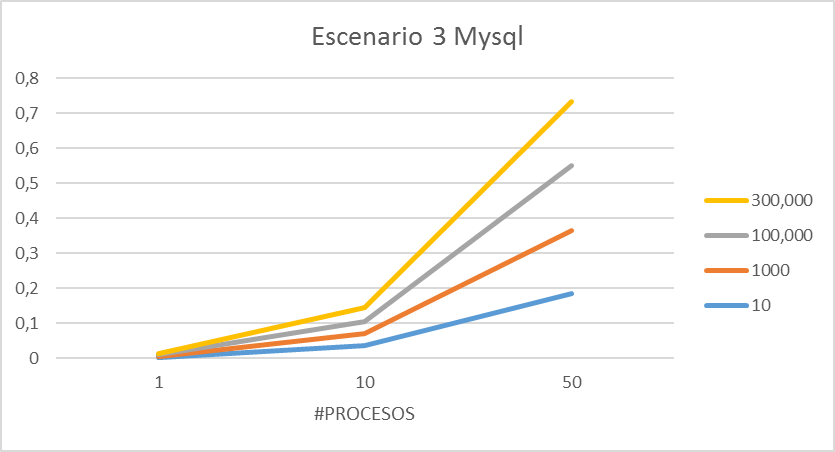


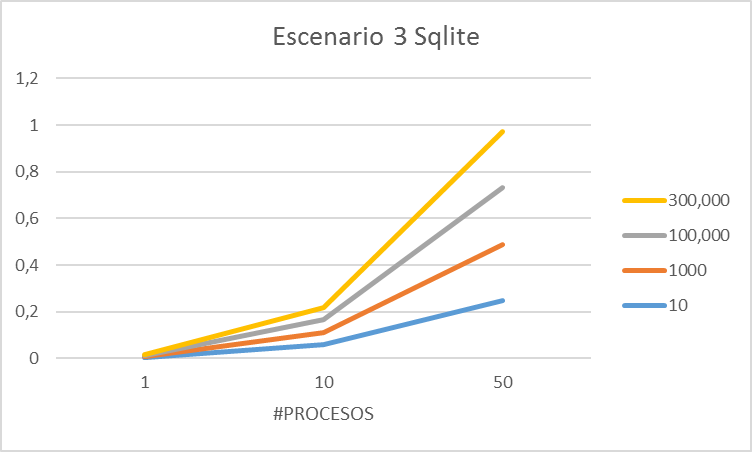


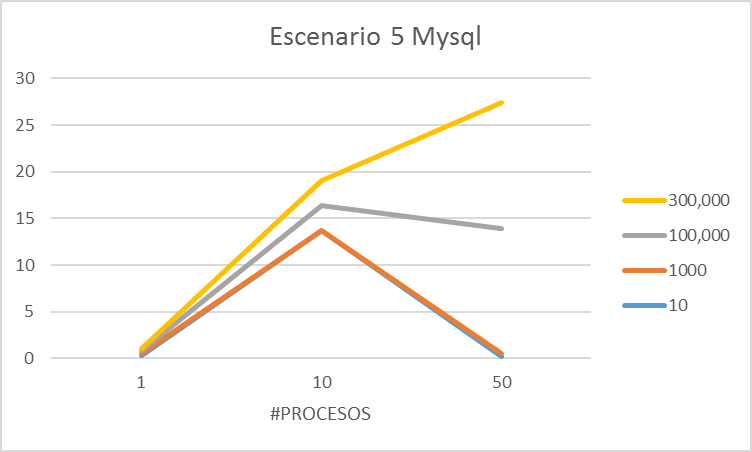
## Escenario 3 y 5: SELECT mono y multi proceso con y sin índice

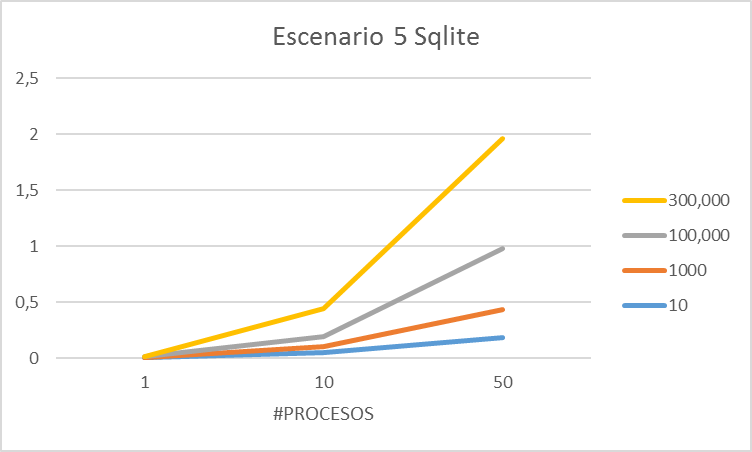
No tiene sentido analizar ambos escenarios por separado. La idea general de estas dos pruebas es observar cómo se comporta el rendimiento cuando existe o no existe un campo indexado en una tabla de una base de datos. Se puede observar que el rendimiento con índice es mucho mejor que aquel en donde no se utiliza el índice. Esto sucede porque, valiéndose del índice, el manejador de datos no tiene por qué explorar toda la tabla para conseguir un récord, dado que ya, revisando los índices puede obtener el mismo en mucho menor tiempo.

La caída de rendimiento se puede observar más claramente en MySQL, en el caso con mayor recurrencias y mayor volumen de queries, la diferencia de tiempo es de hasta 19 segundos para el caso no indexado. Esto es un problema en el mundo de las comunicaciones, dado que este tiempo de espera es en realidad bastante largo. A nivel de concurrencias y sistema operativo ocurre algo parecido al escenario uno. Nuevamente demasiadas concurrencias hacen lenta la tarea en general, pero muy poca también. Existe un punto entre poca y mucha concurrencia en el cual el rendimiento es óptimo.









# Conclusión

Conociendo las diferencias fundamentales entre los manejadores de bases de datos MySQL y SQLite, algunos resultados fueron esperados. Se sabe que ambos manejadores sufren con mucha concurrencia, pero se entendía que SQLite era quien más lo hacía. Sin embargo, en nuestro estudio los tiempos de MySQL son incluso preocupantes (nadie espera casi veinte segundos para obtener un récord. Aun cuando no está indexado).

Referencias

*ACID*. (22 de octubre de 2016). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/ACID

*Consistency*. (24 de octubre de 2016). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Consistency\_(database\_systems)

*Database model*. (23 de octubre de 2016). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Database\_model

*Durability*. (24 de octubre de 2016). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Durability\_(database\_systems)

*Isolation*. (24 de octubre de 2016). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Isolation\_(database\_systems)

*MySQL*. (22 de octubre de 2016). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/MySQL

*Relational Model*. (23 de octubre de 2016). Obtenido de technopedia: https://www.techopedia.com/definition/24559/relational-model-database

*Resumen documental*. (21 de octubre de 2016). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Resumen\_documental

*SQL*. (24 de octubre de 2016). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/SQL

*Sqlite*. (24 de october de 2016). Obtenido de Sqlite: http://sqlite.com/

Tezer, O. (21 de octubre de 2016). *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. Obtenido de Digital Ocean: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

*What you need to know about database performance software*. (25 de octubre de 2016). Obtenido de TechTarget: http://searchdatamanagement.techtarget.com/feature/What-you-need-to-know-about-database-performance-software