RMIT University

COSC2406/2407 – Database Systems

Assignment #1

# Creating Derby and MongoDB databases and implementing a heap file in Java

# Task 1: Derby

## Summary

I created a Java app that creates derby database, reads a file that contains the data to be imported, creates statements and sends them to the database. I created 2 tables: one having all the info about the business and second to have ABNs as help file that was provided with the dataset specifies that there could be more than one. The tables were linked by Business ID field (foreign key) having combination of ID and ABN as primary key. The structure of the tables is:

create table businessNames (id integer not null,

name varchar(200) not null,

status varchar(20) not null,

registerDate date,

cancelDate date,

renewDate date,

stateNumber varchar(10),

state varchar(3),

primary key(id));

create table abns (businessId int not null references businessNames(id),

abn varchar(20) not null,

primary key(businessId, abn));

All the dated were converted from strings to date fields which makes any search by date much more reasonable and easy. The size of varchar fields was taken from the help file provided.

Unfortunately there are no records that hold more than 1 ABN so that I could implement separating them and adding both to the second table (as there’s no example how it would be formatted).

## MyApp

The app connects to the database with the given string. Creates 2 tables and starts iterating through the file separating each by with “\t” delimiter. Changes the format of the date from dd/mm/yyyy to dd.mm.yyyy or sets to null if empty. Replaces any (‘)with doubled (‘’) in the name of the business to escape the character for SQL command. Once there’re 100 records they get inserted into both tables accordingly.

## Issues

I faced a problem of the app being super slow because once it read 1 line from the file it would send insert command to the database. So I decided to collect 100 record (maximum size for insert command) and add them to the database in one go.

## Timing

At first it would take over 25 minutes to finish but after the changes stated above on average it would take 570-600 seconds (9.5 – 10 minutes) to import all 2523932 records to the database.

Take1: 590 seconds

Take2: 579 seconds

Take3: 584 seconds

Take4: 575 seconds

Take5: 583 seconds

# Task 2: MongoDB

## Summary

Mongoimport was a very useful command for importing this kind of data into the database. Firstly I used the command like:

mongoimport --db MyDB --collection business --type tsv --columnsHaveTypes --headerline --file dataset.csv

However, I wanted the dates to be in date format to make it easier in the future to perform a reasonable search. I had 2 choices to do that: create a file that provides the fields’ names and their types or change the header line to provide relevant(specifying the date format to covert with no errors) field types as well. I went with the second choice and changed the first line to:

REGISTER\_NAME.string() Name.string() Status.string() RegistrationDate.date\_ms(dd/MM/yyyy) CancelationDate.date\_ms(dd/MM/yyyy) RenewDate.date\_ms(dd/MM/yyyy) StateNumber.string() State.string() ABN.string()

Also I’ve added –ignoreBlanks flag to the command to skip the empty fields without creating them in the database. So new command looked like this:

mongoimport --db MyDB --collection business --type tsv --columnsHaveTypes --headerline --file text10.csv --ignoreBlanks

After importing the data I decided to get rid of the first field for all the records as it’s the same for all 2523932 records in the file.

db.business.update({}, {$unset: {REGISTER\_NAME:1}}, false, true);

## Issues

I was going to write a script that iterates through the whole database and puts all ABNs into an array, however as I said for Derby as well, there’s no example of the format so I abandoned this idea.

## Timing

I ran the import batch file on my pc at home and on my laptop to get the average time it takes to import the data which was 24-25 seconds. Here is a snapshot of the progress and successful import.

2018-03-31T13:25:48.809+1100 connected to: localhost

2018-03-31T13:25:50.747+1100 [##......................] MyDB.business 18.5MB/215MB (8.6%)

2018-03-31T13:25:53.746+1100 [#####...................] MyDB.business 44.9MB/215MB (20.9%)

2018-03-31T13:25:56.747+1100 [#######.................] MyDB.business 70.6MB/215MB (32.8%)

2018-03-31T13:25:59.747+1100 [##########..............] MyDB.business 97.1MB/215MB (45.1%)

2018-03-31T13:26:02.748+1100 [#############...........] MyDB.business 122MB/215MB (56.8%)

2018-03-31T13:26:05.747+1100 [################........] MyDB.business 147MB/215MB (68.6%)

2018-03-31T13:26:08.746+1100 [###################.....] MyDB.business 173MB/215MB (80.5%)

2018-03-31T13:26:11.746+1100 [######################..] MyDB.business 199MB/215MB (92.7%)

2018-03-31T13:26:13.650+1100 [########################] MyDB.business 215MB/215MB (100.0%)

2018-03-31T13:26:13.651+1100 imported 2523932 documents

Running command to remove REGISTER\_NAME took around 28-29 seconds (28225ms, 30048ms, 27356ms, 29785ms – taken from mongo log file).

Total time taken: ~52.5 seconds.

# Task 3: Implement Heap File in Java

## Heap file structure

Every single record in the file has the following attributes (the first field which contains “BUSINESS NAMES” for all the records is ignored):

* ID(gets created starting from 0)
* Name
* Status
* Registration Date
* Cancelation Date
* Renewal Date
* State Number
* State of Registration
* ABN

ID is an int which is 4 bytes long. Name, State Number and ABN are variable length so the length of the field is stored before it in the heap file. Status is converted to one byte which could be 0 or 1 which means “Deregistered” and “Registered” accordingly. This way we use only 1 byte to store the data instead of storing the whole string. Dates are converted to epoch time, and then converted to long variables, which takes 8 bytes to store. State of registration is stored as a string of size 3 (however only 1 byte could be used to store the state to decrease the output file).

Record structure in a page (example taken from Halil’s post on canvas):

{[ id ][ name length ][ ... name ... ][status][ regDate ][ canDate ][ renewDate ][ stateNum length ][ ... stateNum ... ][ regState ][ abn length ][ ... abn ...]}

Page structure in the heap file:

{[…record…][…record…][…record…][…record…][…record…]………………[numOfRecords]}{[…record…][…record…][…record…][…record…][…record…]……………………[numOfRecords]}…

**dbload output example:**

Page size: 4096

Input file: dataset.csv

Output file: heap.4096

Records loaded: 2523932

Pages used: 47936

Execution time: 7395 milliseconds.

**Some search examples from the app:**

Trying to match: string

4324 AUSTRINGS Registered 2005-05-10 2017-07-15 2014-05-10 BN98138504 NSW 52567019704

9421 STRING IT LIVING ART Registered 2016-02-20 2017-06-30 2017-02-20

22898 RACKETS AND STRINGS Registered 1980-11-01 2017-10-01 2017-10-29 0175375X SA

.

.

.

.

.

.

.

2516427 West Australian string quartet Registered 2018-02-20 2019-02-20

2518404 JOHNSON HV STRINGING SERVICES Registered 2018-02-23 2019-02-23

Matches: 504

Execution time: 2010 milliseconds.

Trying to match: integer

39417 THE INTEGER GROUP Registered 2011-03-08 2017-09-17 2014-03-08 BN98557658 NSW 38077077117

39596 INTEGER IT SUPPORT Registered 2011-09-23 2017-06-09 2017-09-23 BN98596086 NSW

79387 INTEGER IT SOLUTIONS Registered 2005-05-18 2017-06-09 2017-05-18 BN98140520 NSW 97929489155

.

.

.

.

2212163 Integer Data Centre Registered 2017-06-26 2020-06-26

2518752 INTEGER ACCOUNTS Registered 2018-02-27 2021-02-27

Matches: 16

Execution time: 1958 milliseconds.

Trying to match: oh wow

120550 OH WOW BOOKS Registered 2015-11-20 2017-03-30 2016-11-20

120600 OH WOW Registered 2015-11-20 2017-03-30 2016-11-20

1338985 OH WOW! Window Cleaning Home & Office Cleaning Registered 2012-06-06 2018-06-06

Matches: 3

Execution time: 1938 milliseconds.

## Timing

Loading (with page size 4096) took on average 6800-7100 milliseconds.

Search operation time on the same file was on average 1850-2100 milliseconds (increases with increase of matched records)