**Assignment1 – Multiple threading**

**Course: cst8277 Enterprise Application Programming**

**Professor: Rejaul Chowdhury, Stanley Pieda**

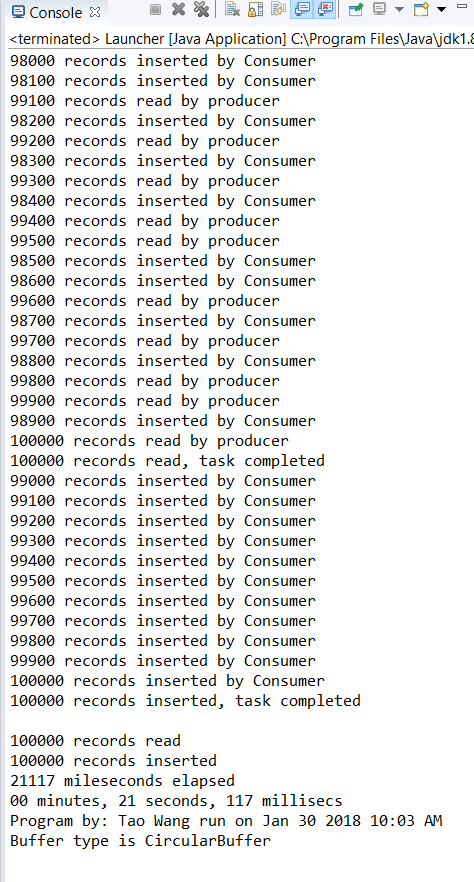
**Lab section:304**

**Student: Tao Wang, 040857654**

**Created date: January 30, 2018**

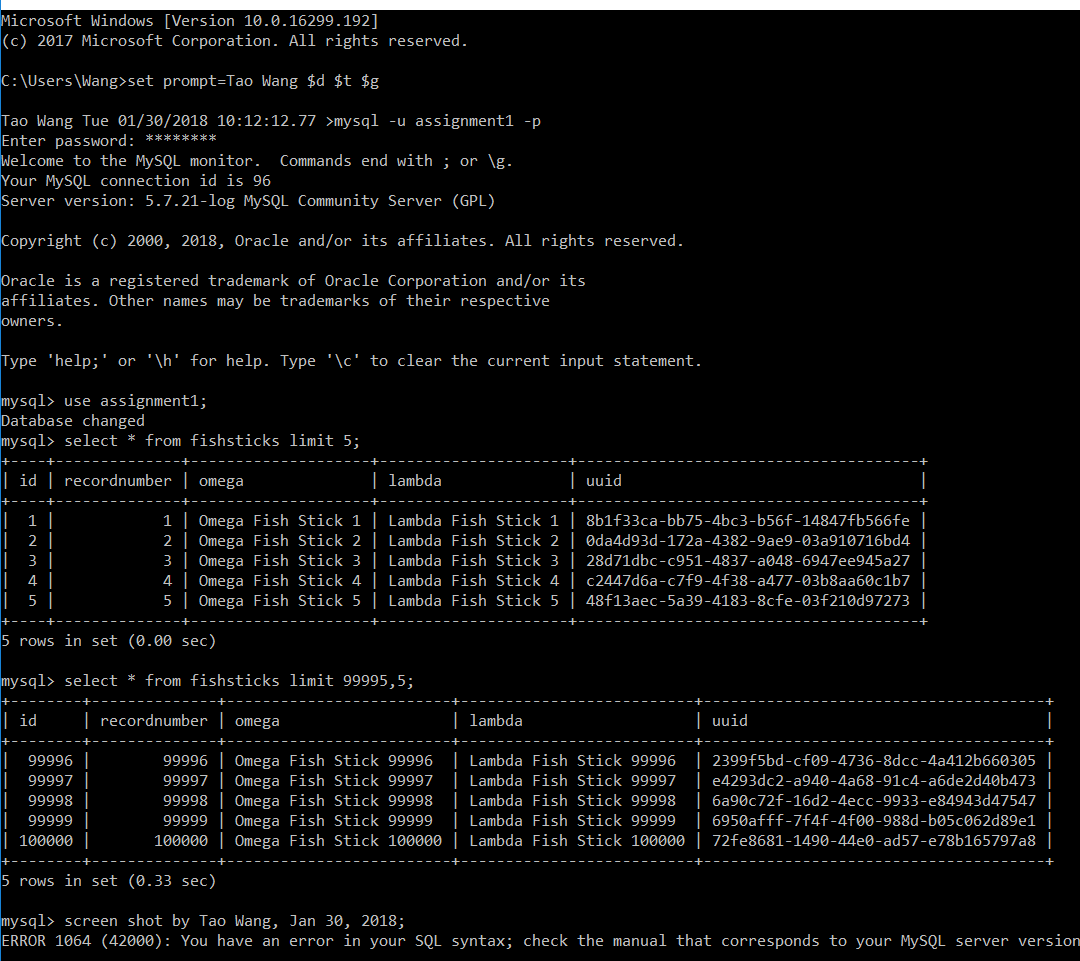
# **Screen shot for program execution**

## BlockingQueueBuffer circular buffer



（The times of the execution between the two types of buffer **are not always the same as the result** shown. Sometime the blocking queue is faster but sometime the circular does. ）

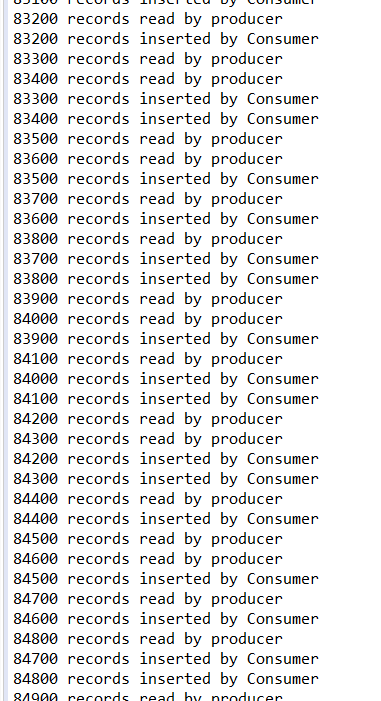
# **Screen shot of database**



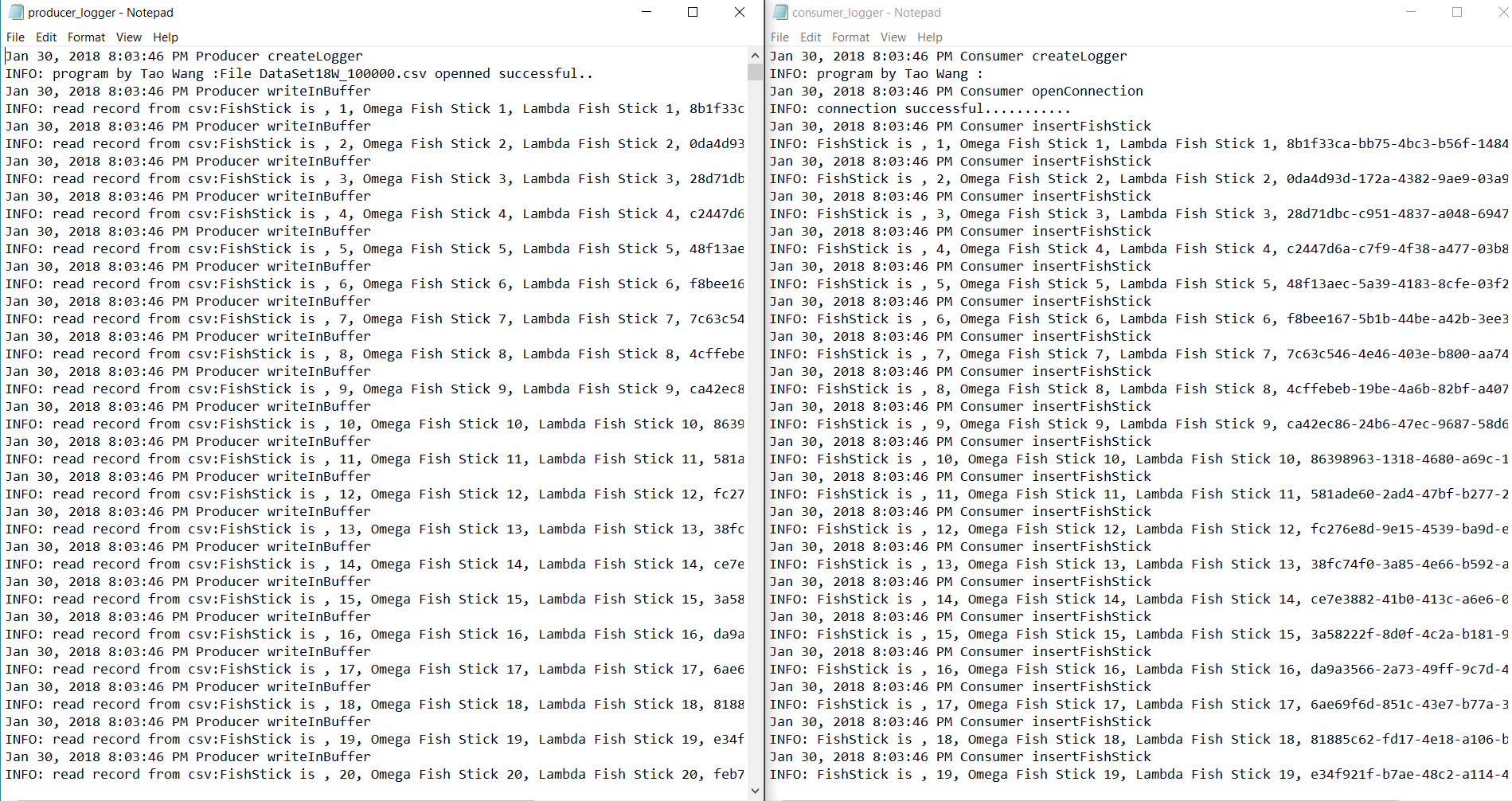
# **Test plan**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Description | Pre-Conditions | Post-Conditions | Test methodology | Expected | Actual | Notes |
| Read and write operations taking place asynchronously | Program has started and it is reading and writing records | File was opened, database was connected to. | Viewing output from producer and consumer screen | Should see read and write messages in tandem.  Should see the different threads in the Jvisualvm.exe | Matches expected. | Test passes |
| the correct records were read | Program has started and it is reading records from the csv file | File was opened and the records were read  correctly | Create Log file of **Producer thread** for tracking the records read from csv file. | Should see the records read in the file  **Producer\_logger.log** match to the records in the csv file. | Matches  expected | Test passes |
| the correct records were inserted | Program has started and it is writing records to database | Database was connected to and records write into the DB. | Create Log file of **Consumer thread** for tracking the records write to the database. | Should see the records write to database in the file  **Consumer\_logger.log** match the records in the database. | Matches  expected | Test passes |
| records were inserted in the same order they were read | Program has started and the records write to database must have the same order as reading from the csv file. | The records of csv were in the same order as the records of database. | Create Log files for comparing the records inserted were in the same order as they read. | Should see the records in the file **producer\_logger.log** have the same order as the records in the file **consumer\_logger.log.** | Matches  expected | Test passes |
| a new thread was used for producer and a new one for the consumer | Program has started and two different threads work for reading and writing the records independently. | The reading task using a thread and writing task use another one. Program will end once all threads killed. | Using the **jvisualvm.exe** to monitor the different threads. | Should see different threads work for reading and writing tasks in the **jvisualvm.exe.** | Matches  expected | Test passes |

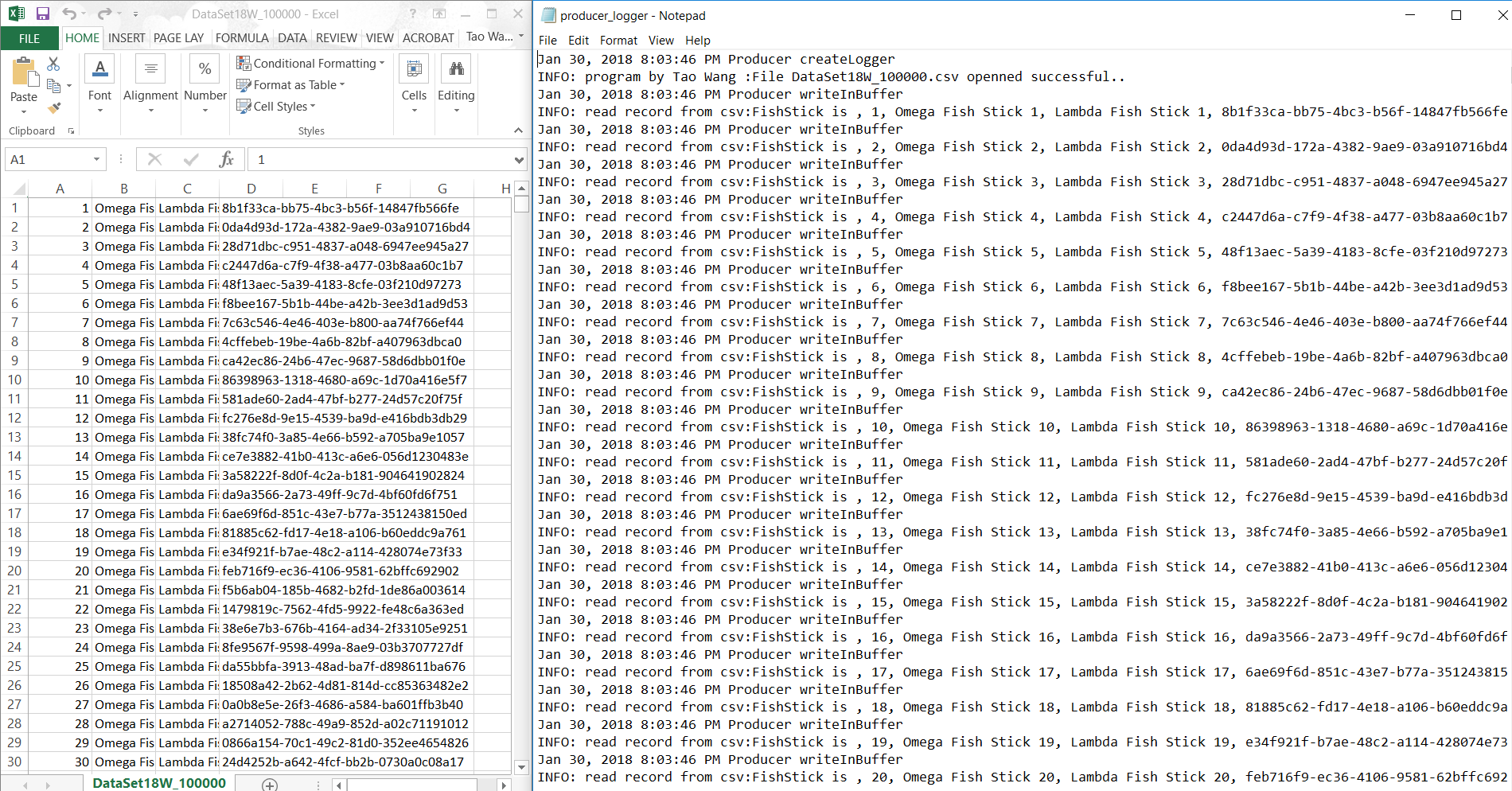
**Screen shot for testing the read and write asynchronously:**



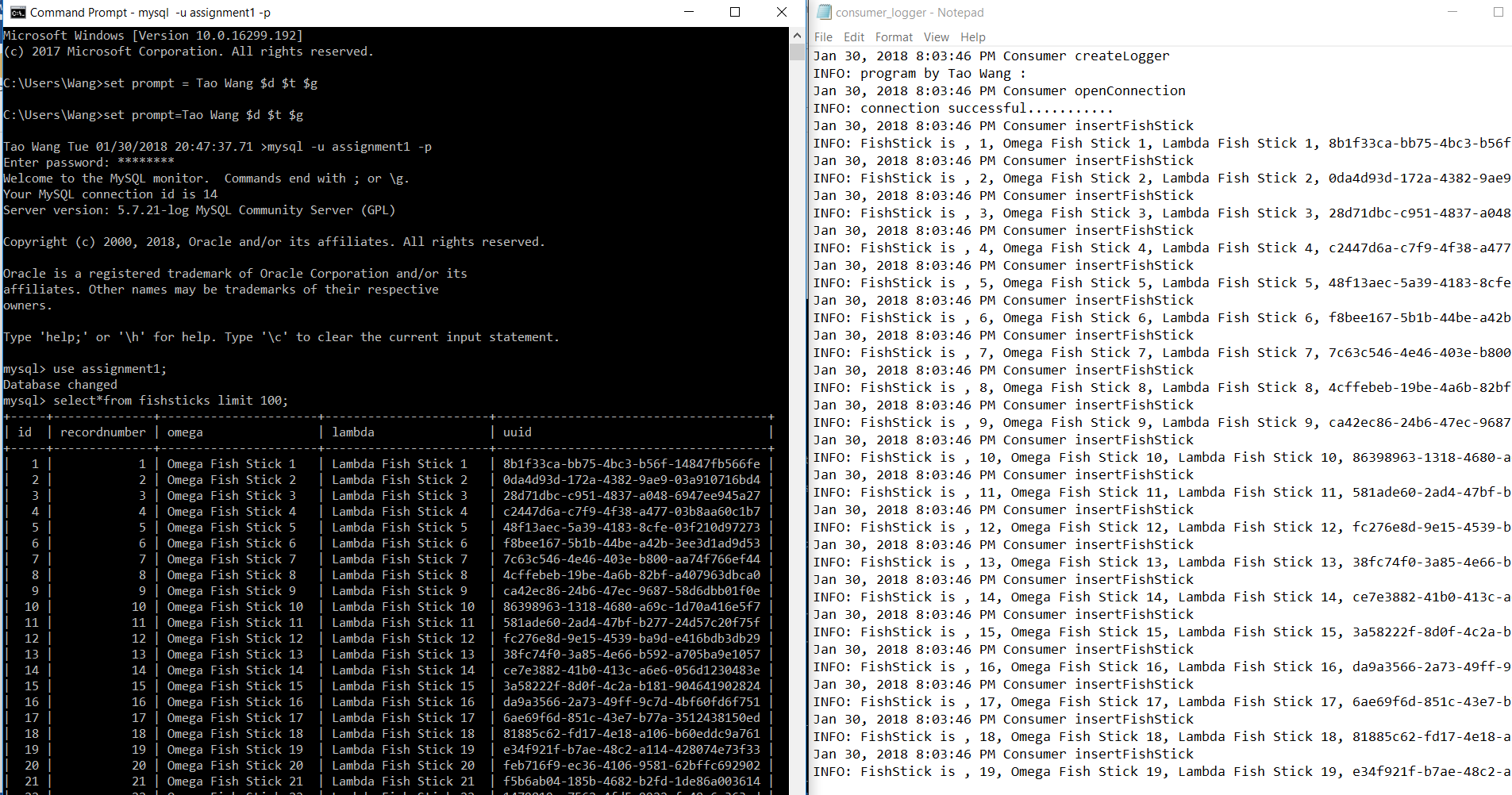
**Screen shots for the two log files to confirm the same order:**



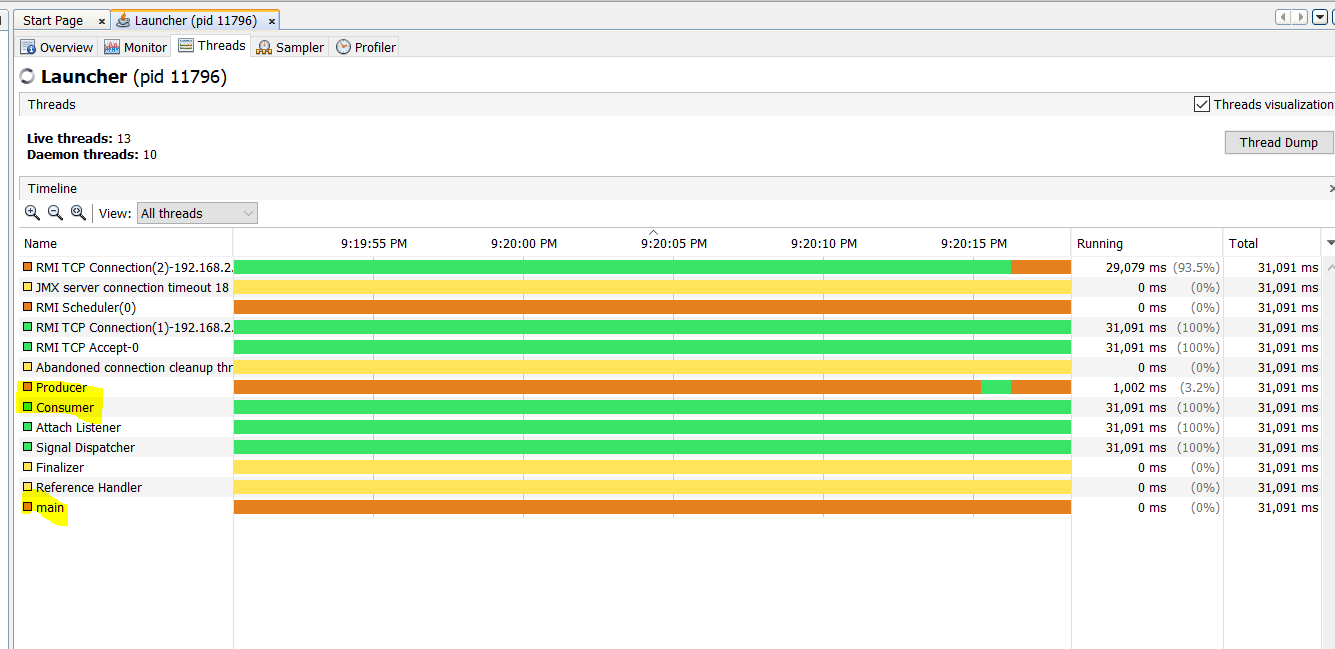
**Screen shots for comparing the csv file and procuder\_logger.log file:**



**Screen shots for comparing the consumer log file and the records in DB:**



**Screen shots for confirming one thread for producer and another one for consumer:**



# **Short answer questions**

## 1. Which implementation of the Buffer was easier to code, the circular buffer using an array and keywords synchronized with methods wait () and notifyAll () or the buffer using the ArrayBlockingQueue?

A: The BlockingQueueBuffer was easier to code because the ArrayBlockingQueue had already been encapsulated in the JDK source code, we could use the class’s functions directly and easily.

2. Ease of implementation aside, what are some advantages and disadvantages of each approach?

For the BlockingQueueBuffer, it is easily to code and easily to use all functionalities of the ArrayBlockingQueue class through a reference variable of the ArrayBlockingQueue in my own buffer class. On the other hand, the encapsulation of the class may be also considered as cons for the extensibility of the program. It is hard to add any new features beyond the ArrayBlockingQueue.

For the circular buffer, it is harder to code than the blocking one because we have to think about how to define and operate a buffer manually include the design of the buffer, the type of container(array), index operation, algorithm, etc. However, the circular buffer is more flexible than the former because we could add any new features we want.

In summary, the blocking buffer is like a car already made by the manufacturer, we could drive it easily and don’t care about how to build it. The circular buffer is like a car has nothing until we install the parts for it. As a result, if the ArrayBlockingQueue could meet all functionalities for a program, my suggestion is the blocking one. Otherwise, the circular buffer is more flexible choice.