The Performance of Different Indexes

Peicheng Wen |

1. **Introduce**

An index allows a query or transaction to retrieve data from a database efficiently. An index is usually related to several columns in a specific table. Indexes consist of one or more keys which are a fancy term for the values we want to look up in the indexes. We can find the specific value via comparing keys to the index.

There are different methods of indexes in database. Non-clustered indexes contain the index keys in sorted order, with the leaf level of the index containing the pointer to the record. There can be more than one non-clustered indexes on one database table. Non-clustered indexes also need extra disk space. Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be sorted in only one order. The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure called a heap.

1. **Benefits of indexes**

The database engine can use indexes to improve performance in many queries. Some of these performance improvements are dramatic. The most obvious use for an index is in finding a record or set of records matching a WHERE clause. Usually, index can improve the performance of SELECT and UPDATE. The DELETE transaction is based on the practical situation. Although we can locate the certain data we want to delete by indexes. We also need to update the indexes after deleting. This may slow down the transaction.

For the WHERE transaction, we did an experiment on testing the performance of B-tree indexes which is the default index in SQL Server 2012. First, we use Python (Figure 1) to create a 100,000 rows \* 24 columns data table (Figure 2) which consists of random integers between 0 to 1000.

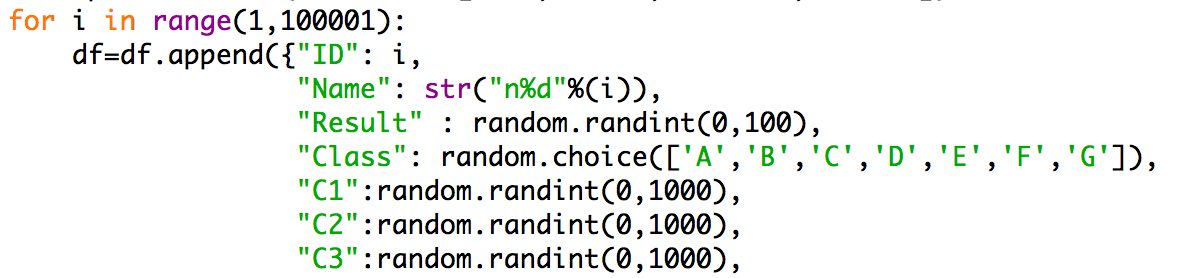


Figure 1 Python script for creating data table

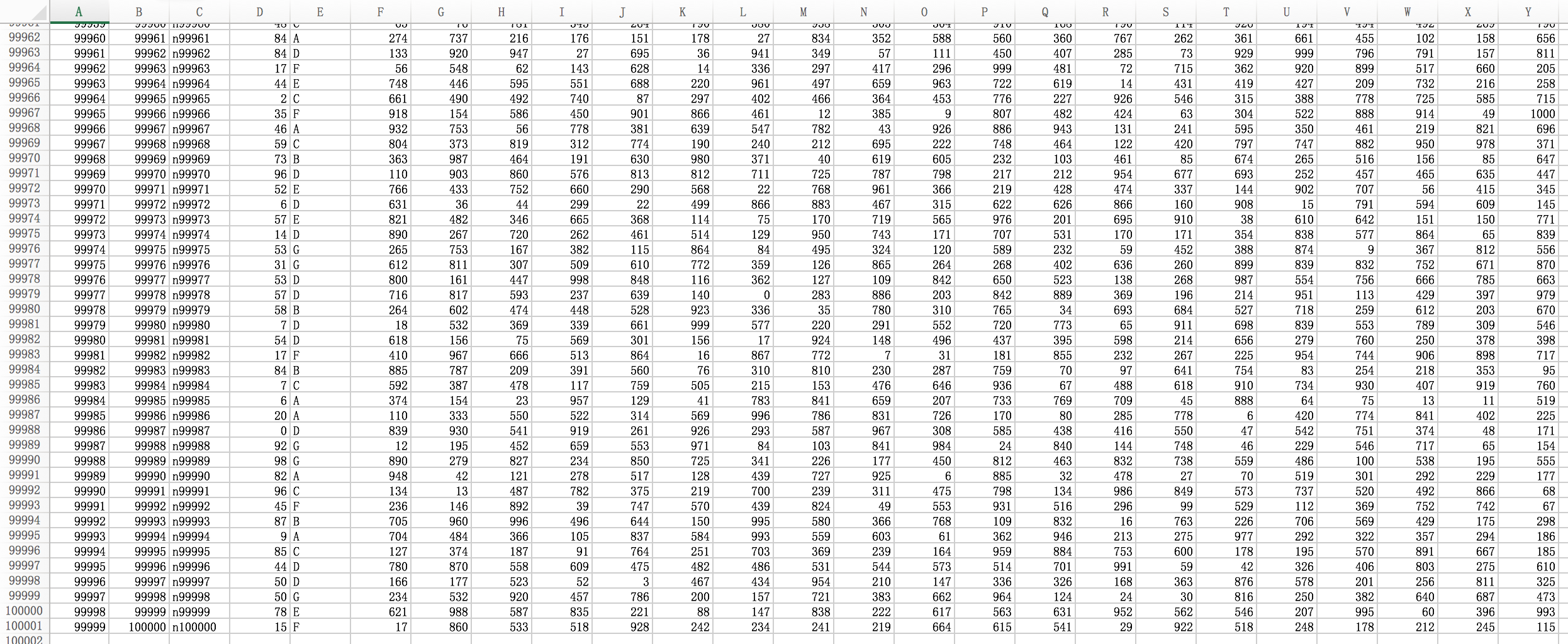


Figure 2 Data Table

For the reason that the difference of 1 transaction between use indexes or not is not obvious at all. We apply a python script (Figure 3) to write 1000 different SELECT transaction with WHERE clause (Figure 4).

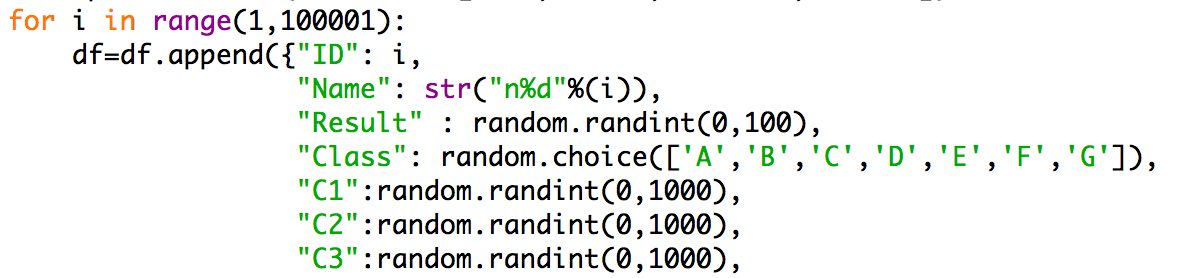


Figure 3 Python script for writing SELECT transaction

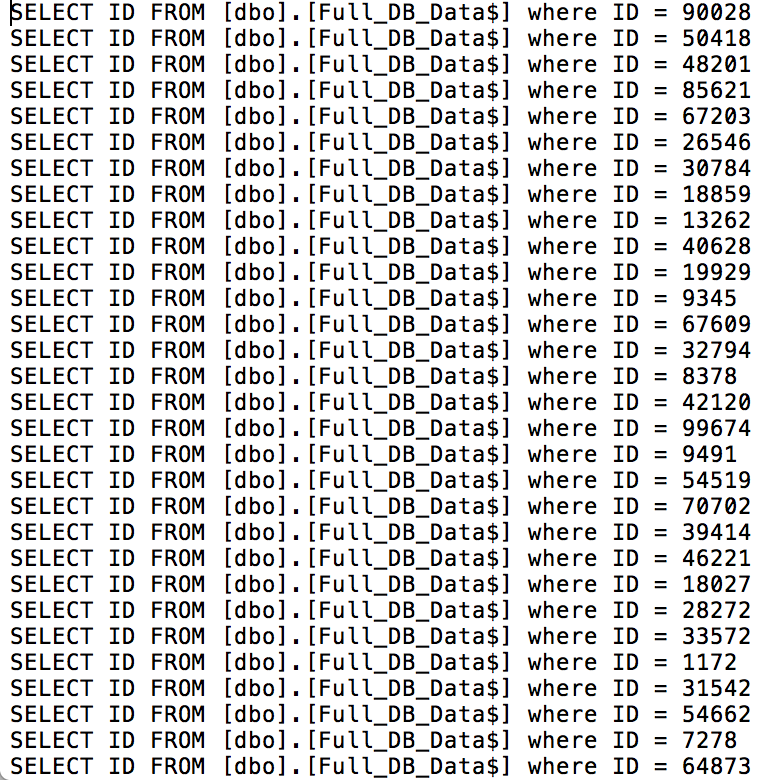


Figure 4 1000 different SELECT transactions

In this experiment, we measure the performance by the Elapsed time. The shorter time the transactions take, the better performance that method has. We get the result which we did not use index as figure 5. And figure 6 is the result with an B-tree clustered index.

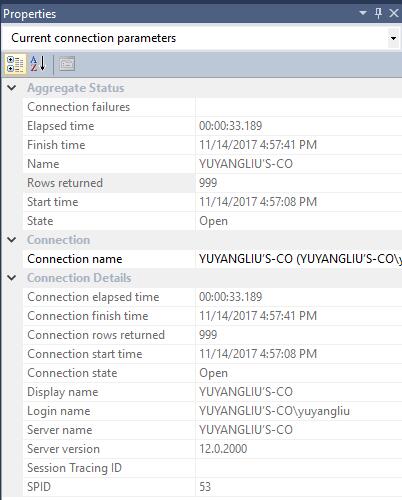


Figure 5 SELECT without indexes

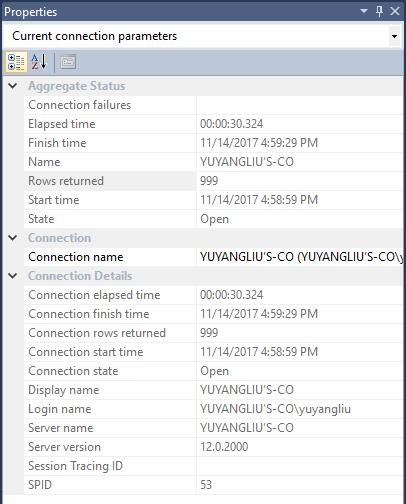


Figure 6 SELECT with an index

Based on the result we have, we can see that without indexes the elapsed time is 33.189. While it is 30.324 with an index in the Figure 6. So, we can know that the index can improve the performance of SELECT transactions with WHERE clause.

1. **Drawbacks of indexes**

The index itself occupies space on disk and memory (when used). So, if space or memory are issues then too many indexes could be a problem. When data is inserted/updated/deleted, then the index needs to be maintained as well as the original data. This slows down updates and locks the tables (or parts of the tables), which can affect query processing.

Our experiment here is based on the INSERT transaction. First, we use a python script (Figure 7) to write 1000 different transactions (Figure 8).

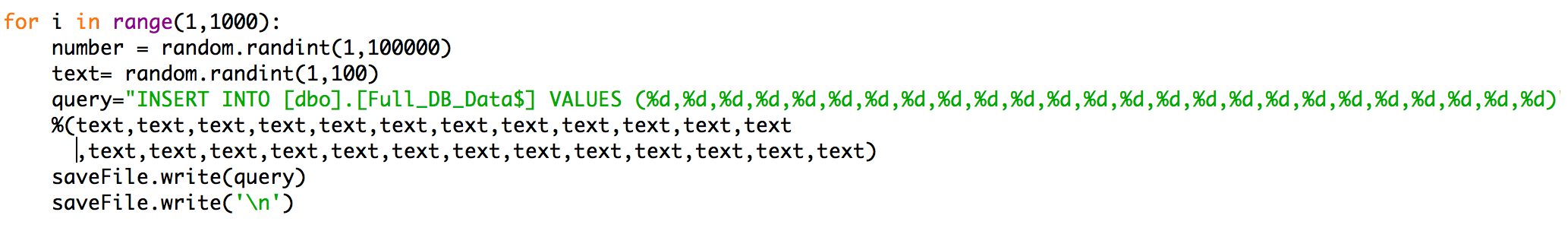


Figure 7 write 1000 different transaction

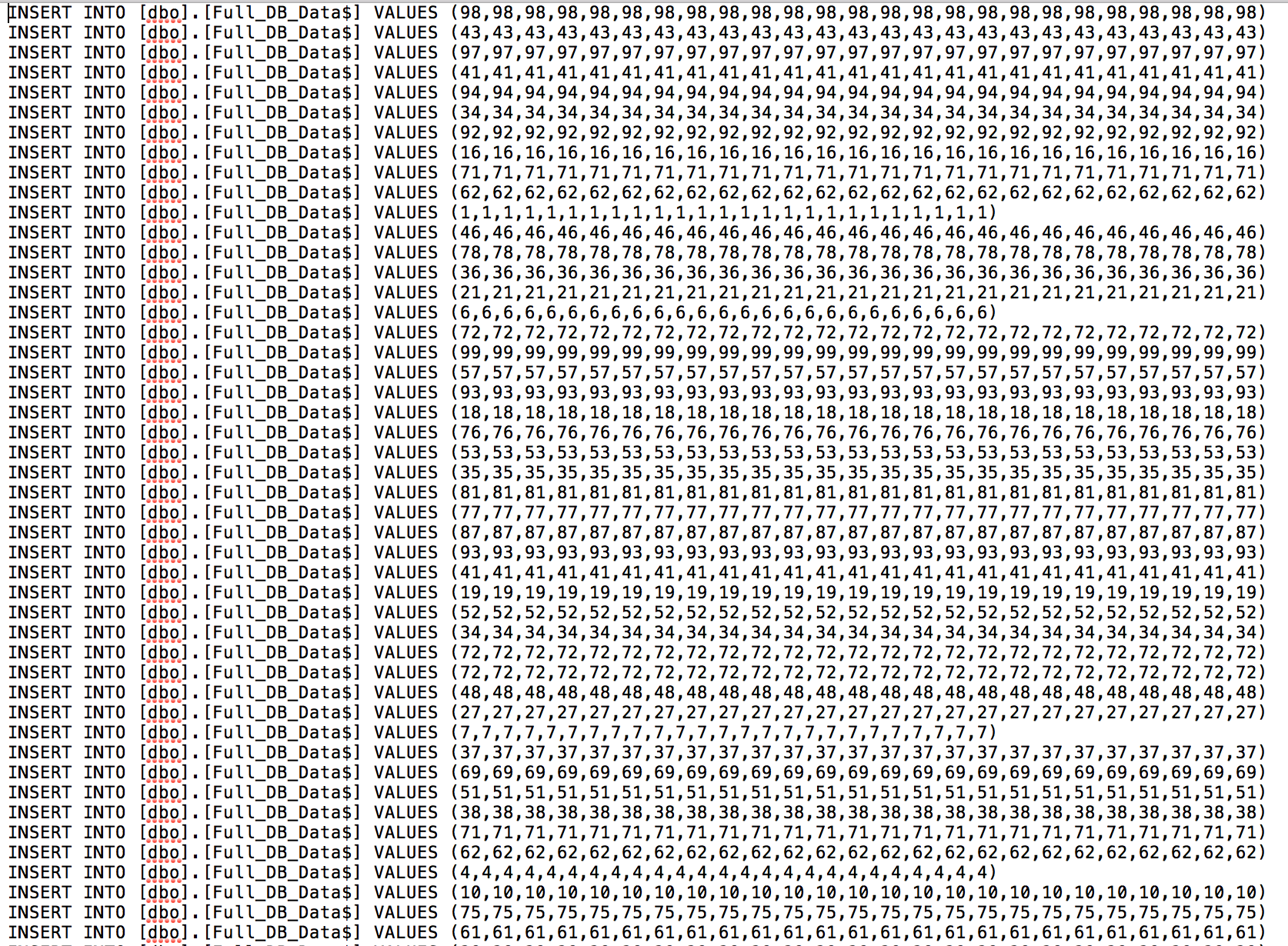


Figure 8 1000 different INSERT transactions

We test these transactions in 3 different cases. First is a database without indexes (Figure 9). Second, we create 5 indexes into the same table we used. We get the result as Figure 10. The third situation is based on the same table but with 25 indexes (Figure 11). We measure the index performance using the Elapsed time as we did in last experiment.

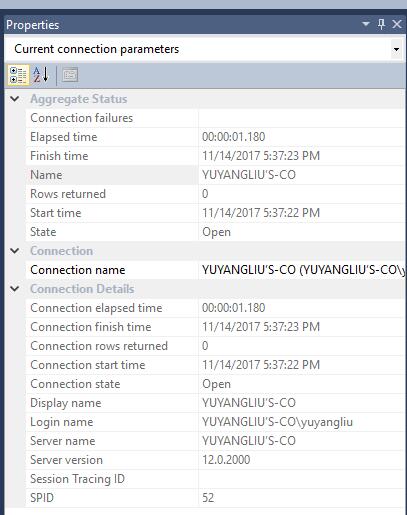


Figure 9 INSERT without indexes

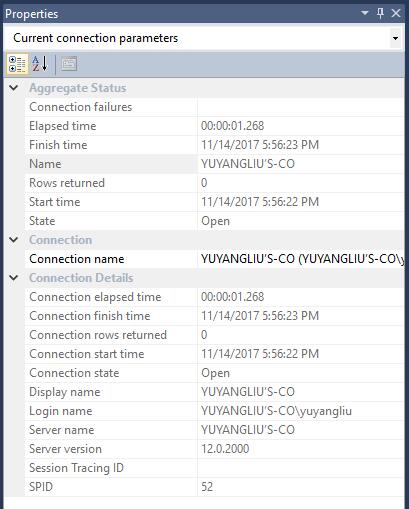


Figure 10 INSERT with 5 indexes

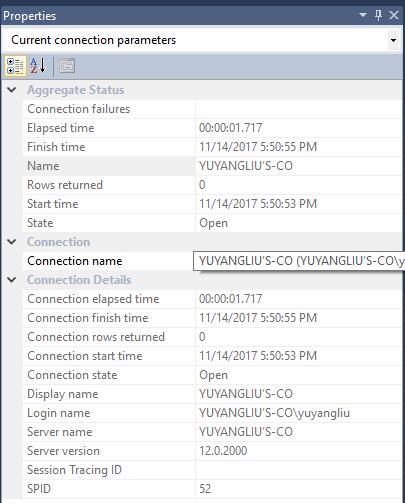


Figure 11 INSERT with 25 indexes

The result we got is accord with we expected. When we have more indexes in the same table, the INSERT transactions are operated more slowly. The main reason is that the INSERT transaction not only insert data into the table, but also update the indexes to make the indexes capable for new data. This can hurt the performance of database.