

Report Findings

1.0 Background

Trim/Content Manager is the application which access the database servers to display the data from the servers. Trim can only connect one database at a time to execute the Structured queries. The database server consists of multiple databases with multiple datasets. The database servers consist of databases of same schema with same attribute values. At present Trim is the 2-tier application. It is where the client can access the database directly. The advantage of this model it is easy to maintain and it is simple. The communication occurs in request and response process between client and database servers. Request from clients follows Tabular data Stream(TDS) protocol and response from the database servers follow the same.

2.0 Introduction

The problem statement is that whether it is possible to access multiple database without changing the client application and database. The solution of the problem is to have a software that run simultaneously that handles the requests of the client application. Solution is to come up with 3-tier application. The purpose is to handle the request of the client application and modify the request. Thereafter, database servers receive the modified request and process that request and send it back to the client.

3-tier application using a middleware can cause the application logic and business rule to be independent such that data are passing from one port to another and processes are getting executed in every tier. Therefore it is important to know what exactly the middleware should do.

3.0 Architecture

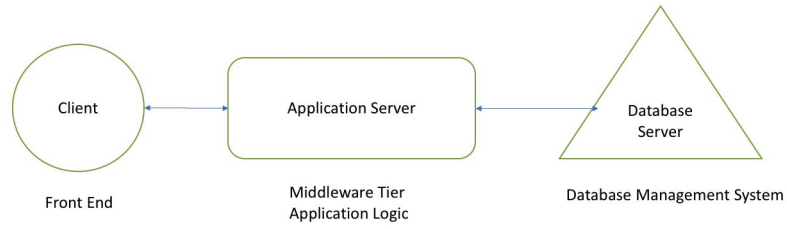


Figure 1: 3-tier Application

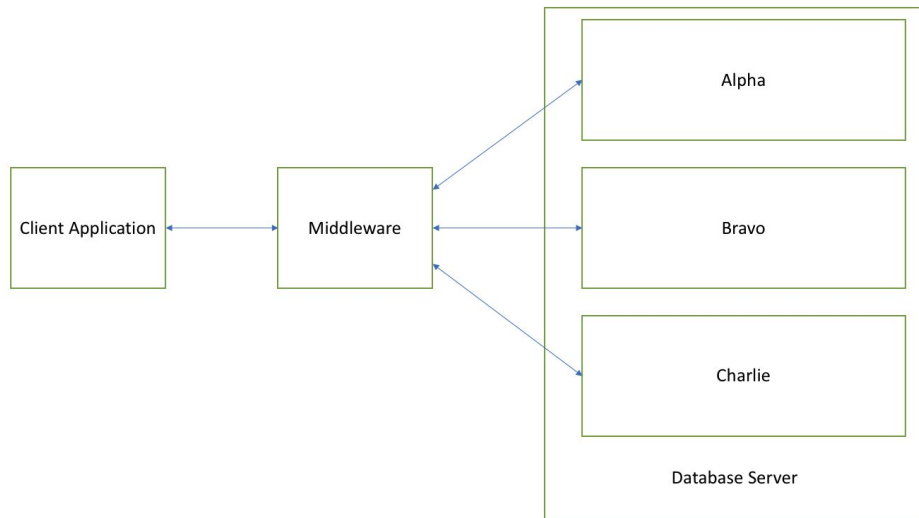


Figure 2: Middleware for different Databases

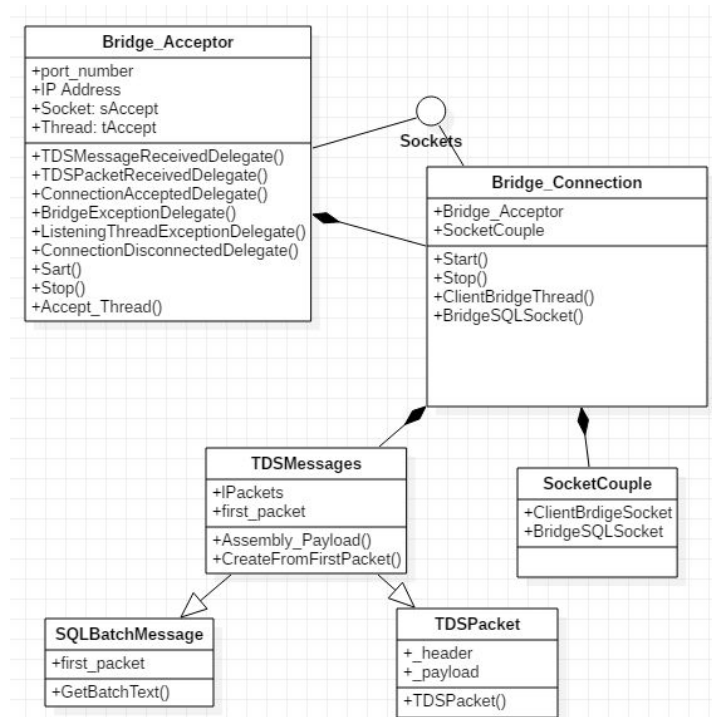


Figure 3: Class Diagram for TDS Bridge Middleware

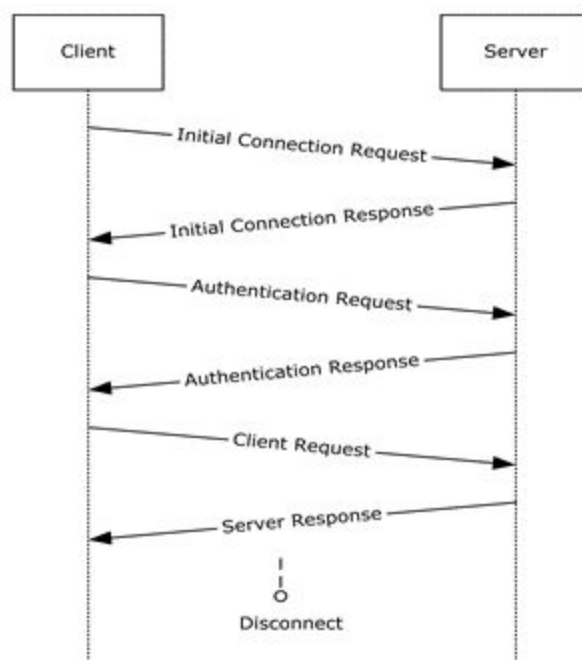


Figure 4: TDS Communication Protocol

4.0 Methodology

4.1 Technology Stack

Due to TRIM's nature of not being open-source and its lack of feature to listen to specific IP address & ports, we believe that our productivity will increase if we were to use a small-scale replica.

As a result, our technology stack includes:

- **SQL Client** - Mini SQL Query <<https://github.com/paulkohler/minisqlquery>>
- **Middleware** - TDS Bridge <<https://github.com/MindFlavor/TDSBridge>>
- **DB Server** - Microsoft SQL Server <techlauncher.icognition.cloud>

4.2 Problems & Ideas

4.2.1 Understanding the Middleware

The first problem we had to face was to understand what the middleware should do. In essence, the middleware should be able to receive incoming queries made by the client, forwards the queries to the database server, and relays the response back to the client. It should also modify the queries as a part of the demultiplexing process (splitting a single query to multiple related queries). TDS Bridge serves as the perfect candidate for this role, due to the fact that it is an open source tool, whilst providing 100% compatibility with the TDS protocol.

4.2.2 Modifying TDS Packets

Modifying TDS packets is required to change incoming queries sent by the client. This serves as the first step to demultiplex an incoming query into multiple queries.

In accordance to the TDS documentation, client and server communication is done through TDS. The TDS packets will then be encapsulated according to the underlying transport protocol it uses, which is TCP in this case since the middleware uses .Net-library sockets as its form of transport channel.

To change the queries, we needed to figure out where the SQL statements sit within the TDS message. In reference to the TDS documentation, the SQL statements appear to be encoded in Unicode, thus we decided to decode the entire TDS bytestream to Unicode, giving the following result:

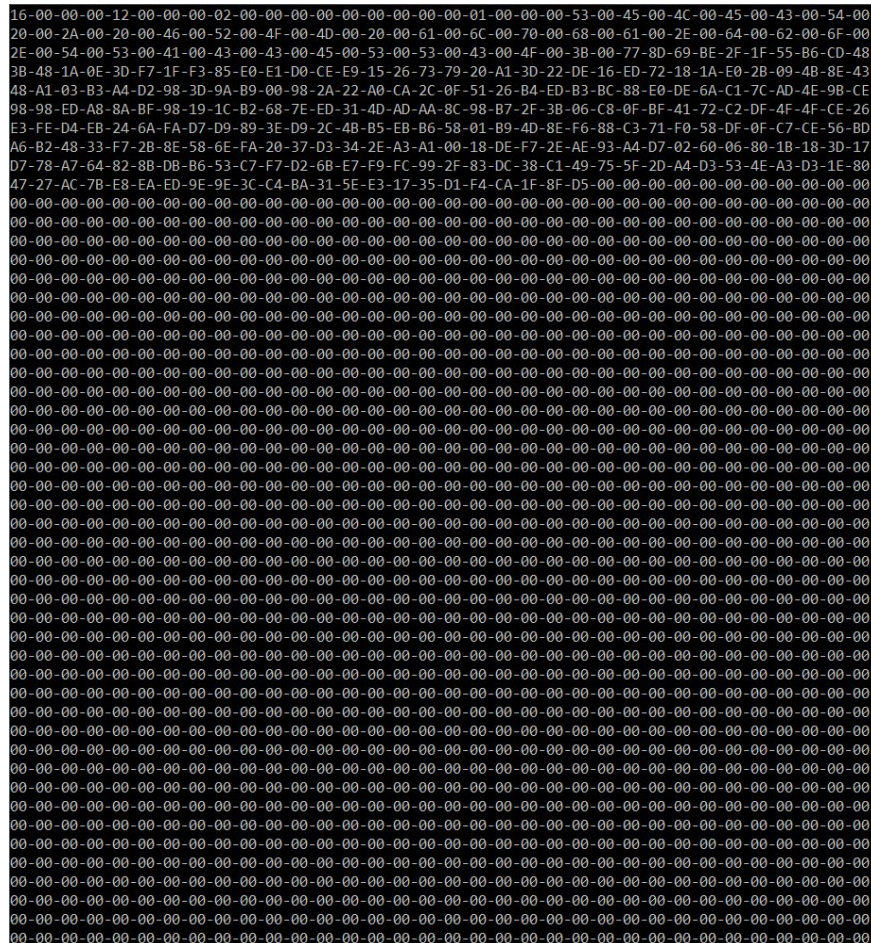


Figure 5: TDS bytestream in Unicode

Next, we converted the above Unicode to a human readable and modifiable form - a string, giving the following result:

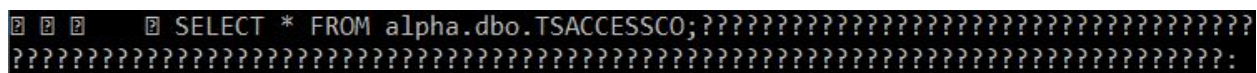


Figure 6: Unicode to string conversion

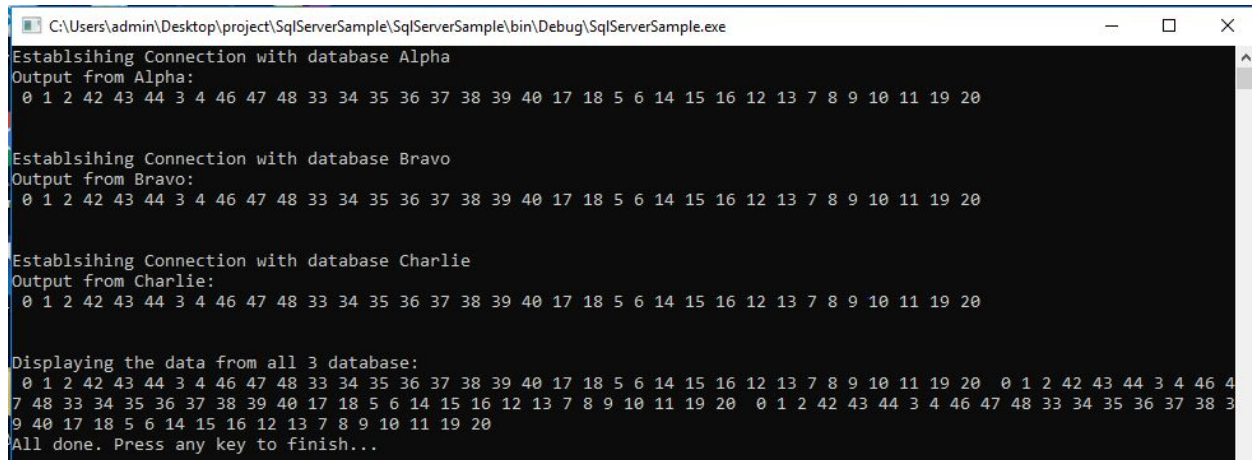
Lastly, we attempted to modify the query ***"SELECT * FROM alpha.dbo.TSACCESSCO;"*** to ***"SELECT * FROM bravo.dbo.TSACCESSCO;"***, and convert the entire string back to unicode before sending it across the TCP socket. This, however, yielded undefined behaviours.

We later discovered that attempting to convert non-text bytestreams to a string format may cause certain control-characters and data which is not representable by string to be lost. In the case of the incoming TDS message, only the SQL statements within that message is represented in Unicode and also string compatible (***section 2.2 of the TDS documentation***). The rest of the data within the packet, however, are not Unicode-to-string compatible, which explains the garbage characters found in figure 6 after the conversion.

Our future plan to tackle this issue would be to only convert the SQL statements data within the entire TDS message to string, to ensure safe conversions without any data loss. This can be done by finding the specific offset of where the SQL statements reside within the packet, since network protocol packet headers are always fixed. This, however, requires an extensive research on how an SQL Batch TDS Message is formatted, as well as an immense amount of time debugging the formatted packets.

4.2.3 What the Middleware Should do

Our aim was to build an interface that listens to the client application. However, once the queries are sent by the client, the queries are encapsulated within a message protocol format, which carries several problems stated in the above section.



```
C:\Users\admin\Desktop\project\SqlServerSample\SqlServerSample\bin\Debug\SqlServerSample.exe
Establsihing Connection with database Alpha
Output from Alpha:
0 1 2 42 43 44 3 4 46 47 48 33 34 35 36 37 38 39 40 17 18 5 6 14 15 16 12 13 7 8 9 10 11 19 20

Establsihing Connection with database Bravo
Output from Bravo:
0 1 2 42 43 44 3 4 46 47 48 33 34 35 36 37 38 39 40 17 18 5 6 14 15 16 12 13 7 8 9 10 11 19 20

Establsihing Connection with database Charlie
Output from Charlie:
0 1 2 42 43 44 3 4 46 47 48 33 34 35 36 37 38 39 40 17 18 5 6 14 15 16 12 13 7 8 9 10 11 19 20

Displaying the data from all 3 database:
0 1 2 42 43 44 3 4 46 47 48 33 34 35 36 37 38 39 40 17 18 5 6 14 15 16 12 13 7 8 9 10 11 19 20 0 1 2 42 43 44 3 4 46 4
7 48 33 34 35 36 37 38 39 40 17 18 5 6 14 15 16 12 13 7 8 9 10 11 19 20 0 1 2 42 43 44 3 4 46 47 48 33 34 35 36 37 38 3
9 40 17 18 5 6 14 15 16 12 13 7 8 9 10 11 19 20
All done. Press any key to finish...
```

Figure 7: Multiplexed Output

In the figure above, we have made a pseudo query demultiplexing scenario where 3 separate SQL queries were sent to 3 different databases, with the response multiplexed together.

This is what our middleware is supposed to do. However, this functionality lies in the application layer, before the queries are converted into packets, which allows modifications. Modifying packets is not as trivial since the packet format has to be kept constant.

5.0 Conclusion

In conclusion, we have yet to come up with a concrete solution on demultiplexing queries. However, we have hypothesised several methods on making query modifications possible. We believe that if we could successfully modify incoming client query statements from the TDS message and relay it to the database server without data loss, demultiplexing queries can most certainly be possible as well.