CHAPTER 4
ANALYSIS AND DESIGN

4.1 Analysis

This project aims to analyze and compare the message broker performance results on the E-Ticketing System. This project focuses on speeding up the process of sending ticket data. When a visitor has completed a transaction on the E-Ticketing System, the system will process the payment in general. It will give the visitor a ticket code. Visitors will later use the ticket code to enter the tourist attractions that have been targeted. The ticket code will be unique and will be automatically generated immediately after the payment has been successful. After the ticket code has been generated, the ticket code will be stored in the database and recorded as a used ticket code. This process will take a long time because each process must wait until the previous process has been completed. This process will make the response time long, and if the process lasts too long, it can make visitors lose their patience. One thing that can overcome this is to use a message broker.

The message broker is capable of sending data without having to wait for the previous process to complete. This process is because the message broker works asynchronously, which is a condition where processes can run simultaneously. What message brokers can do on the E-Ticketing System is to send the ticket code that has just been created to visitors, along with the process of entering data into the database. This process can speed up response time so that visitors immediately receive a ticket code that has been generated previously.

This project aims to analyze and compare the message broker's data delivery performance on the E-Ticketing System. Each of the message brokers will calculate the latency of receiving data based on the time the data is sent. The data stored in the database include data delivery speed (in milliseconds), the amount of data sent in one unit of time, CPU usage and memory usage. The data will be stored using the PostgreSQL.

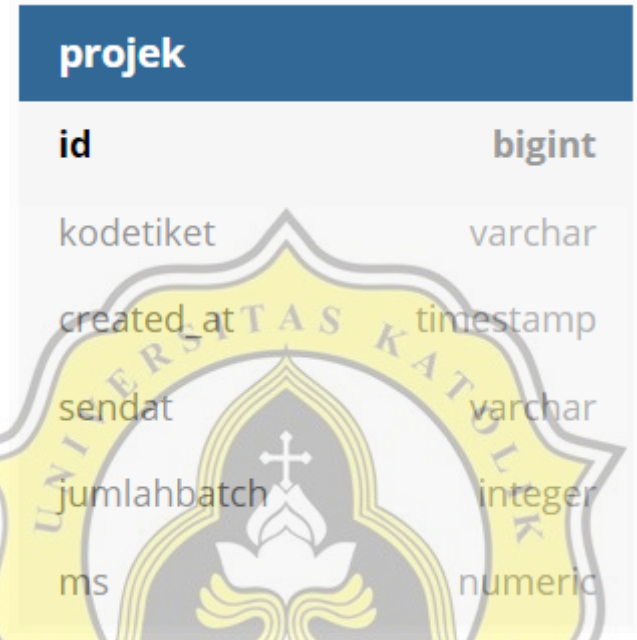

Figure 4.1: Projek Design Database

Above is the design of the project's database. The database has several columns such as id, kodetiket, created_at, sendat, jumlahbatch and ms. Kodetiket is a column to store the ticket code that the system has generated. Created_at is used to store the time when the system created the ticket. Sendat is used to store the name of the message broker the message is Sent. Jumlahbatch is used to store the amount of data sent at one time. Ms is used to calculate latency when entering ticket code data into the database.

Later the ms column in this database will be used to prove the importance of using the broker process in the E-Ticketing System. The speed of latency in entering data into the database will affect the user's response time. By using a message broker, the system can respond to a ticket code without having to wait until the insert process into the database is complete.

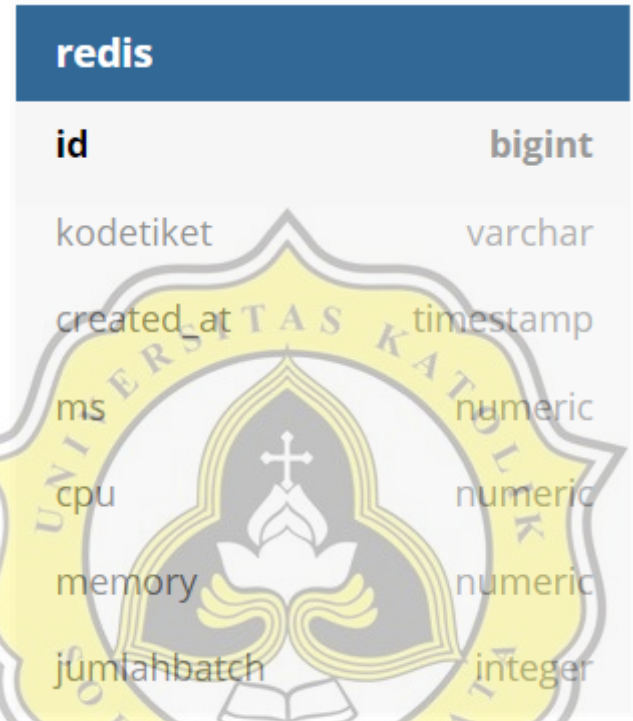

Figure 4.1 : Redis Design Database

Above is the design of the Redis database. The database design in question is the database design to accommodate the data received by Redis. The type of database used is Postgresql version 13.0. This database has columns id, kodetiket, created_at, ms, CPU, memory, and batch count. Kodetiket is used to store ticket data that Redis have received. Created_at is when the data has been received by Redis and will be stored in the database. Ms is used to store the latency speed (ms) of receiving data from the publisher. CPU is used to save CPU usage when consuming data. Memory is used to save memory usage when the process of using the message broker. Jumlahbatch is used to classify data based on the amount of data sent at one time.

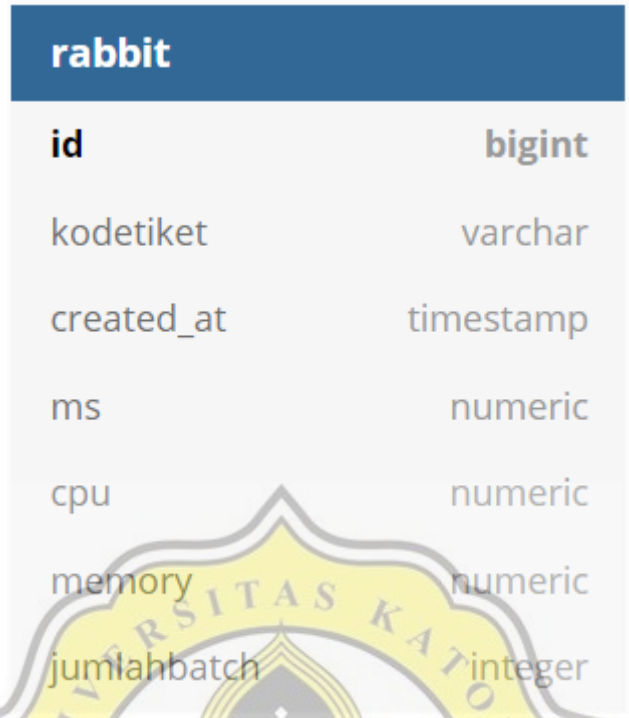

Figure 4.1 : Rabbit Design Database

Above is Rabbit's database design. Rabbit's database design is the same as Redis' database. The use of each column also has the same thing as kodetiket used to store the sent ticket code, ms to record the latency (ms) of data transmission, CPU to store CPU performance, memory to deviate memory performance, and jumlahbatch used to classify data based on the amount of data that sent in one time.

To prove that message brokers can be used in different databases, in this study, each message broker will use a different database to store data.. Created_at is used to store when the program created the data. The ticket code is used to store the ticket code sent by the sender. Ms is a column to store latency in ms. CPU is a column to store CPU Usage in % units. Memory is a column to store Memory Usage in % units. The number of batches is to store how much data is sent in one unit of time. The batch count is also very useful to make it easier to measure the average value based on the amount of data sent.

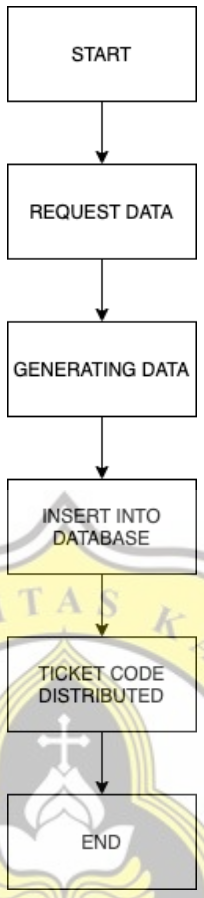

Figure 4.1 : E-Ticketing Sample Flowchart

Below is a simple E-Ticketing System flowchart diagram without using a message broker. When the system receives a request, the system will generate a ticket code or booking code. After that, the code will be stored in the database. After the code saving process is complete, the system will send the code to the customer.

Systems with such a process flowchart above have some drawbacks. One disadvantage of this system is when the system is getting a lot of requests, and it will lead the process of data storage into the database will belong. This issue causes the acceptable response time will take some time because the process of delivery of ticket code must wait until the data storage process is completed.

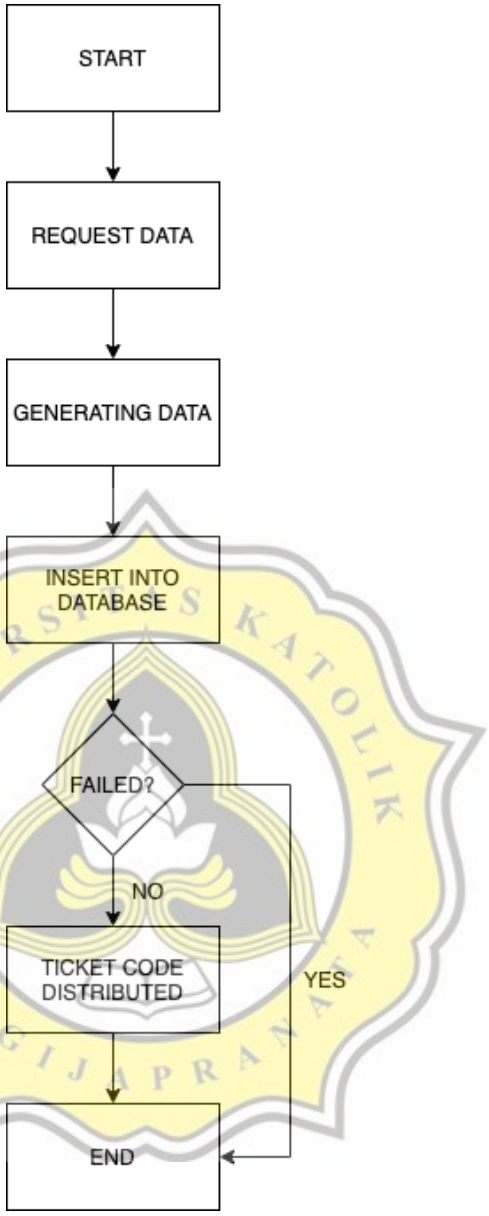

Figure 4.1 : E-Ticketing Sample Worst Case Flowchart

The other drawback is that when the data insertion process fails, the next process will stop, making visitors repeat the transaction process. If this happens to tourist attractions that have high visitor traffic, it will cause long queues of visitors. One alternative way to overcome this is to use a message broker.

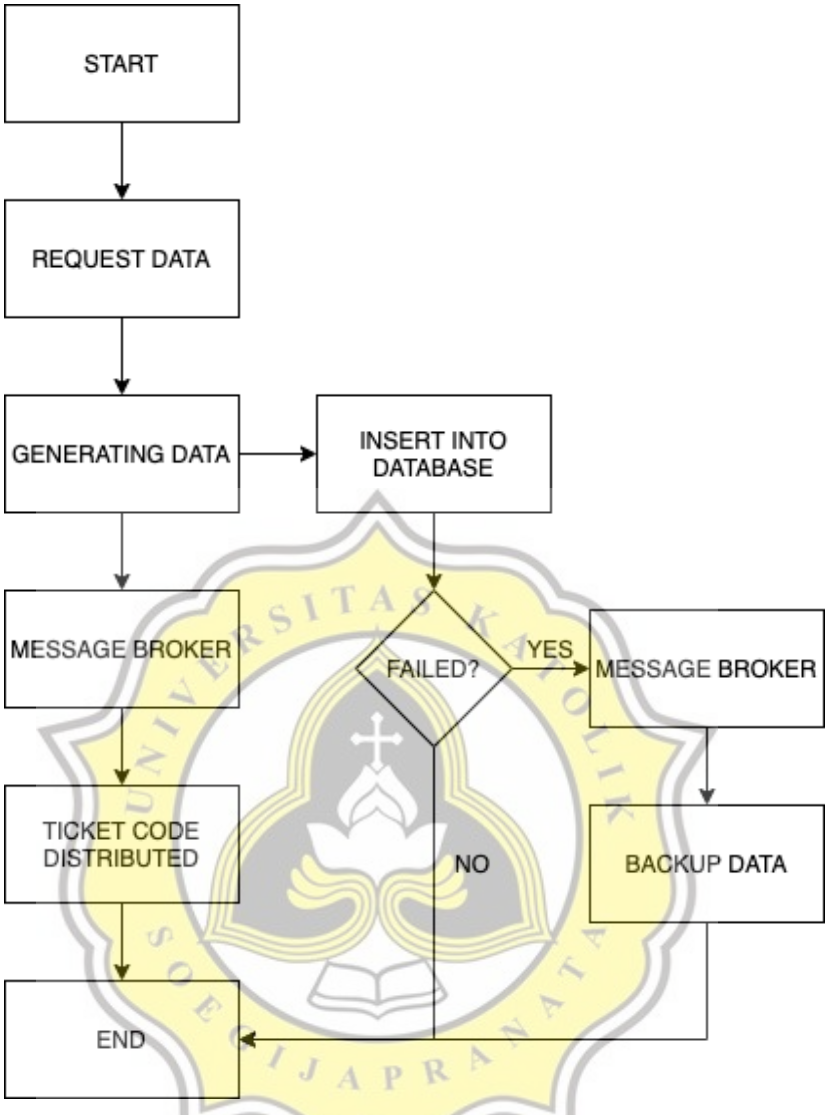

Figure 4.1 : E-Ticketing System with Message Broker

Message brokers can overcome both problems mentioned above. The first problem is performance. With the message broker's asynchronous performance, the message broker can solve this problem by sending a ticket code generated without having to wait for the code saving process to complete. This difference will be seen in the amount of data sent at one time, where it takes a long time to store the data in the database.

For example, below is the result of the latency performance of entering data into the database. These results are obtained by testing the Postgres database to store data for a certain period and with a certain amount of data. When the database inserts many data, the process will take a long time, and from the table below, there is a high latency difference between 1000 data and 10000 data.

Table 4.1: Table Average Insert Latency (ms)

Data Amount	Average Latency (ms)
1	1.2848537
100	4.58617131
1000	55.2547529
10000	2562.2

Therefore, a message broker is needed so that the data transmission process does not have to wait until the saving process is completed first. The message broker will split the process and send the data. The use of message brokers can reduce the response time of a process.

In addition, the message broker can also overcome cases when the process of storing data fails. When the database fails when saving data, the message broker can save the data into memory and trying to save the data again when the database is available.

4.2 Design

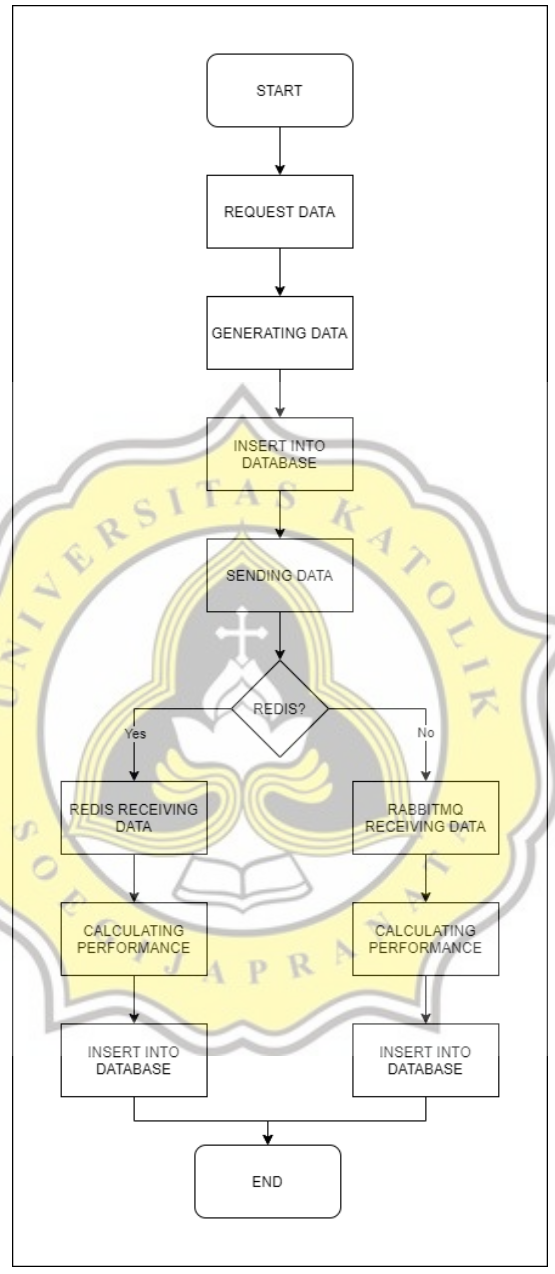

Figure 4.2 : Flowchart Process

Above is a flowchart of how this program process takes place. The first is to input the request. The intended request is the amount of data that will be sent per unit of time. Variations in the amount of time used in this testing there are 4 types, namely 1, 100, 1000,

and 10000 data. Then the system will generate data in the form of a unique ticket code. Then the data will be entered into the Postgresql database that has been provided. Then the data will be sent via message broker. The message broker will send the data without having to wait for the process of entering data into the database to complete. Then the data will be sent via one of the message brokers that have been determined.

After the data is received by the message broker, the system will perform a performance calculation. Performance that will be calculated is CPU usage, Memory, and latency during delivery. The calculated data will then be stored in the database.