APPENDIX

1. PUBLISH CODE

IMPORT PACKAGE

```
package main

import (
   "database/sql"
   "encoding/json"
   "fmt"
   "log"
   "strconv"
   "strings"
   "time"

   "github.com/go-redis/redis"
   _ "github.com/lib/pq"
   "github.com/streadway/amqp"
)
```

POSTGRES CONNECTION

```
func PostgresConn() (*sql.DB, error) {
   postgres, err := sql.Open("postgres", "host=localhost port=5432 user=postgres
   password=password dbname=projek sslmode=disable")
   if err != nil {
       log.Println("PostgresConn error : ", err.Error())
       return nil, err
   }
   return postgres, nil
}
```

REDIS CONNECTION

```
func RedisConn() *redis.Client {
   redis := redis.NewClient(&redis.Options{
       Addr:     "localhost:6379",
       Password: "",
       DB:       0,
   })
   return redis
}
```

RABBIT CONNECTION

```
func RabbitConn() *amqp.Connection {
   rabbit, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
   if err != nil {
       log.Println("Error RabbitMQ Conn :", err.Error())
   }
   return rabbit
}
```

CREATE STRUCT

```
type Data struct {
   KodeTiket  []string
   TimeCreate string
}
```

CALL POSTGRES CONNECTION

```
postgres, err := PostgresConn()
if err != nil {
      log.Println("Error Postgres Conn :" + err.Error())
}
```

CLOSE POSTGRES CONNECTION AFTER BEING USED

```
defer postgres.Close()
```

CALL REDIS CONNECTION

```
redis := RedisConn()
```

CLOSE REDIS CONNECTION AFTER BEING USED

```
defer redis.Close()
```

CALL RABBIT CONNECTION

```
rabbit, err := RabbitConn().Channel()
if err != nil {
	log.Println("Error Rabbit Conn :" + err.Error())
}
```

CLOSE RABBIT CONNECTION AFTER BEING USED

```
defer rabbit.Close()
```

DECIDING AMOUNT OF BATCH DATA, TIME INTERVAL, AND TIME START

```
now := time.Now()
var dataAmount int
runTime := 40 * time.Minute
intervalTime := 1 * time.Second
fmt.Println("Input Batch Data :")
fmt.Scanln(&dataAmount)
```

BEGIN LOOPING TO CREATE DATA

```
run log.Println("Running...")
for {
```

STOP LOOPING IF RUNNING TIME ALREADY 40 MINUTES

```
if time.Since(now) >= runTime {
      log.Println("Stop...")
      break
}
```

SAVE THE TIME WHEN DATA WAS CREATED

```
tm := time.Now()
timecreate := tm.Format("2006-01-02 15:04:05")
```

CREATE VARIABLE TO SAVE DATA TO BE STORED IN POSTGRES

```
inCondition := ""
var listInterface []interface{}
```

CREATE VARIABLE TO SAVE DATA TO BE SEND IN MESSAGE BROKER

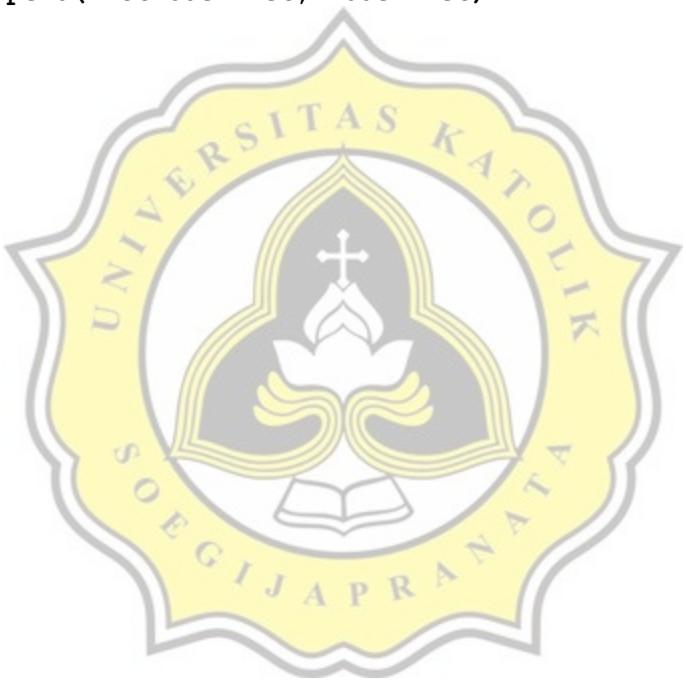
```
var listKodeTiket []string
```

LOOPING AS MUCH AS THE DATA WANT TO BE CREATE

```
for i := 0; i < dataAmount; i++ {
```

CREATE THE TICKET CODE AND SAVE IT TO VARIABLE

```
kodeTiket := fmt.Sprintf("TIKET-"+timecreate+"-"+"%06d", i)
listKodeTiket = append(listKodeTiket, kodeTiket)
```



INPUTTING THE REQUIRED DATA TO BE STORED IN DATABASE

```
listInterface = append(listInterface, timecreate, kodeTiket, "RABBIT", batchData)
	if inCondition != "" {
		inCondition += ", "
	}
	inCondition += "(?,?,?,?)"
}
```

RECORD THE TIME THE DATA WILL BE SENT

```
go func() {
timesend := time.Now().Local().Format("2006-01-02 15:04:05.000000 MST")
```

INPUT THE DATA TO BE SENT

```
sendData := Data{
	KodeTiket:  listKodeTiket,
	TimeCreate: timesend,
}
```

CONVERT THE DATA INTO BYTES

```
dudu, _ := json.Marshal(sendData)
```

PUBLISH THE DATA USING RABBITMQ

```
go func() {
	err = rabbit.Publish(
		"",
		"projek",
		false,
```

```
            false,
            amqp.Publishing{
                  ContentType: "text/plain",
                  Body:        dudu,
            })
      if err != nil {
            log.Println("Error Rabbit Publish :" + err.Error())
      }
}()
```

PUBLISH THE DATA USING REDIS

```
go redis.Publish(redis.Context(), "projek", dudu)

}()
```

SAVE THE DATA INTO DATABASE

```
go func() {

start := time.Now()

query := "INSERT INTO tiket(created_at, kodetiket, sendto, jumlahbatch) VALUES
" + inCondition

query = ReplaceSQL(query, "?")

_, err = postgres.Exec(query, listInterface...)
```

PRINT POSTGRES QUERY ERROR IF EXISTS

```
if err != nil {

      log.Println("Error Postgres Query :" + err.Error())

}
```

SAVE THE INSERT LATENCY INTO DATABASE

```
end := time.Since(start).Milliseconds()

query2 := "UPDATE tiket SET ms = $1 WHERE kodetiket IN ('" +
strings.Join(listKodeTiket, "','") + "')"

_, err = postgres.Exec(query2, end)
```

PRINT POSTGRES QUERY ERROR IF

```
if err != nil {

      log.Println("Error Postgres Query :" + err.Error())

      }

}()
```

SET TIME TO DELAY

```
time.Sleep(runTime)
}
```

2. REDIS SUBSCRIBE CODE

IMPORT PACKAGE

```
import (
	"database/sql"
	"encoding/json"
	"log"
	"math"
	"strconv"
	"strings"
	"time"

	"github.com/go-redis/redis"
	_ "github.com/lib/pq"
	"github.com/shirou/gopsutil/cpu"
	"github.com/shirou/gopsutil/mem"
)
```

POSTGRES CONNECTION

```
func PostgresConn() (*sql.DB, error) {
  postgres, err := sql.Open("postgres", "host=localhost port=5432 user=postgres
  password=password dbname=projek sslmode=disable")
  if err != nil {
      log.Println("PostgresConn error : ", err.Error())
      return nil, err
  }
  return postgres, nil
}
```

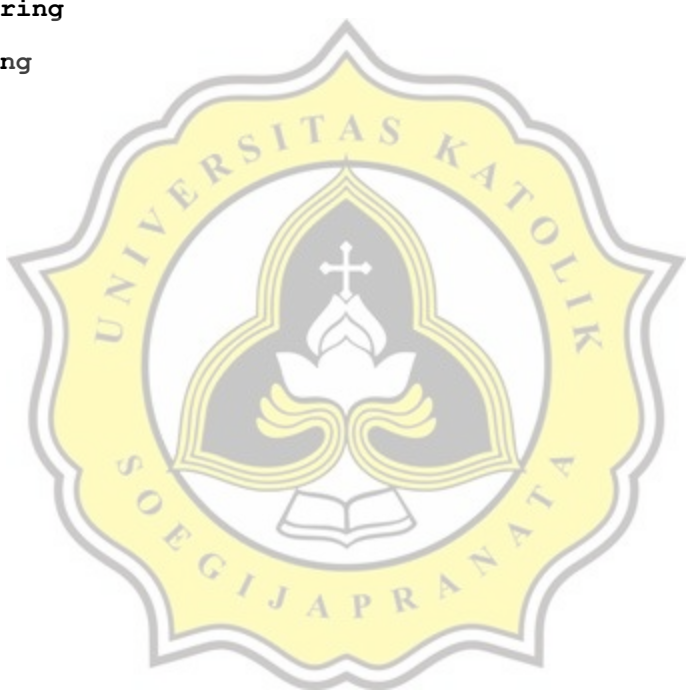
REDIS CONNECTION

```
func RedisConn() *redis.Client {
  redis := redis.NewClient(&redis.Options{
```

```
        Addr:     "localhost:6379",
        Password: "",
        DB:       0,
    })
    return redis
}
```

CREATE STRUCT

```
type Data struct {
    KodeTiket  []string
    TimeCreate string
}
```



CALL POSTGRES CONNECTION

```
postgres, err := PostgresConn()
if err != nil {
   log.Println("Error Postgres Conn :" + err.Error())
}
```

CLOSE POSTGRES CONNECTION AFTER BEING USED

```
defer postgres.Close()
```

CALL REDIS CONNECTION

```
redis := RedisConn()
```

CLOSE REDIS CONNECTION AFTER BEING USED

```
defer redis.Close()
```

DEFINE REDIS SUBCRIBE CHANNEL WITH KEY

```
redissub := redis.Subscribe(redis.Context(), "projek")
ch := redissub.Channel()
```

MAKE A VARIABLE FOREVER AS MARK

```
forever := make(chan bool)
```

START LOOPING EACH MESSAGE

```
log.Println("Running...")
go func() {
```

```
for msg := range ch {
```

RECEIVE THE DATA

```
var val []byte = []byte(msg.Payload)
var data Data
```

UNMARSHAL THE DATA RECEIVE INTO STRUCT

```
json.Unmarshal(val, &data)
```

CHANGE DATA FORMAT OF THE TIME WHEN THE MESSAGE IS SENT

```
timecreate, _ := time.Parse("2006-01-02 15:04:05.000000 MST", data.TimeCreate)
```

CALCULATING THE MEMORY USAGE

```
memory, _ := mem.VirtualMemory()
memoryUsage := int(math.Ceil(memory.UsedPercent))
```

CALCULATING THE CPU USAGE

```
cpu, _ := cpu.Percent(time.Second, false)
cpuUsage := int(math.Ceil(cpu[0]))
```

LOOPING FOREACH DATA KODETIKET

```
go func() {
   for _, kode := range data.KodeTiket {
```

CREATE VARIBALE TO SAVE THE DATA LATER

```
var listInterface []interface{}
inCondition := ""
kodetiket := kode
```

CALCULATING LATENCY SINCE THE DATA RECEIVED

```
ms := time.Since(timecreate).Milliseconds()
timeStampNow := time.Now().Format("2006-01-02 15:04:05")
```

INPUT ALL THE VALUE INTO VARIABLE

```
listInterface = append(listInterface, timeStampNow, kodetiket, ms,
  len(data.KodeTiket), cpuUsage, memoryUsage)
if inCondition != "" {
```

```
   inCondition += ", "
}
inCondition += "(?,?,?,?,?,?)"
```

QUERY TO INSERT INTO DATABASE

```
query := `INSERT INTO tiket(created_at, kodetiket, ms, jumlahbatch, cpu, memory)
   VALUES ` + inCondition
```

CONVERT "?" TO PARAMETER VARIABLE

```
query = ReplaceSQL(query, "?")
```

EXECUTING THE QUERY

```
_, err = postgres.Exec(query, listInterface...)
```

PRINT POSTGRES QUERY ERROR IF EXISTS

```
if err != nil {
   log.Println("Error Rabbit Consume : " + err.Error())
}
```

BACK TO VARIABLE FOREVER

```
   <-forever
}
```

FUNCTION TO CONVERT "?" TO PARAMETER POSTGRES

```
// ReplaceSQL ...
func ReplaceSQL(old, searchPattern string) string {
   tmpCount := strings.Count(old, searchPattern)
   for m := 1; m <= tmpCount; m++ {
       old = strings.Replace(old, searchPattern, "$"+strconv.Itoa(m), 1)
   }
   return old
}
```

3. RABBIT CONSUMER CODE

IMPORT PACKAGE

```
package main
import (
   "database/sql"
   "encoding/json"
   "log"
   "math"
   "strconv"
```

```
    "strings"
    "time"
    _ "github.com/lib/pq"
    "github.com/shirou/gopsutil/cpu"
    "github.com/shirou/gopsutil/mem"
    "github.com/streadway/amqp"
)
```

POSTGRES CONNECTION

```
// Postgres
func PostgresConn() (*sql.DB, error) {
  postgres, err := sql.Open("postgres", "host=localhost port=5432 user=postgres
  password=password dbname=projekrabbit sslmode=disable")
  if err != nil {
      log.Println("PostgresConn error : ", err.Error())
      return nil, err
  }

  return postgres, nil
}
```

RABBIT CONNECTION

```
// Rabbit
func RabbitConn() *amqp.Connection {
  rabbit, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
  if err != nil {
      log.Println("Error RabbitMQ Conn :", err.Error())
  }
  return rabbit
}
```

CREATE STRUCT

```
type Data struct {
  KodeTiket  []string
  TimeCreate string
}
```

MAIN FUNCTION

```
func main() {
```

CALL POSTGRES CONNECTION

```
postgres, err := PostgresConn()
if err != nil {
   log.Println("Error Postgres Conn :" + err.Error())
}
```

CLOSE POSTGTRES CONNECTION AFTER BEING USED

```
defer postgres.Close()
```

CALL RABBIT CONNECTION

```
rabbit, err := RabbitConn().Channel()
if err != nil {
   log.Println("Error Rabbit Conn :" + err.Error())
}
```

CLOSE RABBIT CONNECTION AFTER BEING USED

```
defer rabbit.Close()
```

CALL RABBIT CONSUMER

```
msg, err := rabbit.Consume(
   "projek",
   "",
   true,
   false,
   false,
   false,
   nil,
)
```

IF RABBIT CONNECTION FAILED PRINT THE ERROR

```
if err != nil {
   log.Println("Error Rabbit Consume : " + err.Error())
}
```

MAKE A VARIABLE FOREVER AS MARK

```
forever := make(chan bool)
```

START LOOPING EACH MESSAGE

```
log.Println("Running...")
go func() {
   for d := range msg {
```

RECEIVE THE DATA AND UNMARSHAL IT TO STRUCT

```
var data Data
json.Unmarshal(d.Body, &data)
```

CHANGE DATA FORMAT OF THE TIME WHEN THE MESSAGE IS SENT

```
timecreate, _ := time.Parse("2006-01-02 15:04:05.000000 MST", data.TimeCreate)
```

CALCULATING THE MEMORY USAGE

```
memory, _ := mem.VirtualMemory()
memoryUsage := int(math.Ceil(memory.UsedPercent))
```

CALCULATING THE CPU USAGE

```
cpu, _ := cpu.Percent(time.Second, false)
cpuUsage := int(math.Ceil(cpu[0]))
```

LOOPING FOREACH DATA KODETIKET

```
go func() {
  for _, kode := range data.KodeTiket {
```

CREATE VARIBALE TO SAVE THE DATA LATER

```
var listInterface []interface{}
inCondition := ""
kodetiket := kode
```

CALCULATING LATENCY SINCE THE DATA RECEIVED

```
ms := time.Since(timecreate).Milliseconds()
timeStampNow := time.Now().Format("2006-01-02 15:04:05")
```

INPUT ALL THE VALUE INTO VARIABLE

```
listInterface = append(listInterface, timeStampNow, kodetiket, ms,
  len(data.KodeTiket), cpuUsage, memoryUsage)
if inCondition != "" {
```

```
   inCondition += ", "
}
inCondition += "(?,?,?,?,?,?)"
```

QUERY TO INSERT INTO DATABASE

```
query := `INSERT INTO tiket(created_at, kodetiket, ms, jumlahbatch, cpu, memory)
   VALUES ` + inCondition
```

CONVERT "?" TO PARAMETER VARIABLE

```
query = ReplaceSQL(query, "?")
```

EXECUTING THE QUERY

```
_, err = postgres.Exec(query, listInterface...)
```

PRINT POSTGRES QUERY ERROR IF EXISTS

```
if err != nil {
   log.Println("Error Postgres Query :" + err.Error())
}
```

BACK TO VARIABLE FOREVER

```
<-forever
```

FUNCTION TO CONVERT "?" TO PARAMETER POSTGRES

```
// ReplaceSQL ...
func ReplaceSQL(old, searchPattern string) string {
   tmpCount := strings.Count(old, searchPattern)
   for m := 1; m <= tmpCount; m++ {
       old = strings.Replace(old, searchPattern, "$"+strconv.Itoa(m), 1)
   }
   return old
}
```

Report #13389057

Introduction Background In the E-Ticketing System, a high enough performance is needed to overcome the large number of visitors in a tourist spot. However, the system must also pay attention to the use of existing computer resources, so an efficient system is needed to avoid long queues of visitors and make system performance heavy. In this case the author conducted a study to reduce the performance of the computer and speed up the system process by using Message Broker as middleware to transmit data. Message Broker itself is able to run asynchronously so that data transmission will be faster because there is no need to wait until the previous process is complete. In this study a message broker will be used to send ticket codes, in order to reduce system response time. In addition, this study will use two different message brokers, namely Redis and RabbitMQ. 1 2 3 4 5 6 Redis stands for Remote Dictionary Server which uses memory as a database, cache and message broker. While RabbitMQ is a message broker based on Queue. This study will

z