Proyecto de Simulación

Paulo Barrantes, André Flasterstein, Fabián Álvarez

24 de junio de 2018

Escuela de Ciencias de la Computación e Informática

Universidad de Costa Rica.

Índice general

Capítulo 1

Manual de instalación

El proyecto de Simulación se realizó utilizando Java JDK Versión 8 y usando para la interfaz gráfica del usuario JavaFX con una librería llamada JFoenix para cambiar los estilos de los componentes de JavaFX obteniendo un resultado similar a los botones entre otras cosas de Google Material Design.

Importar proyecto en IntelliJ

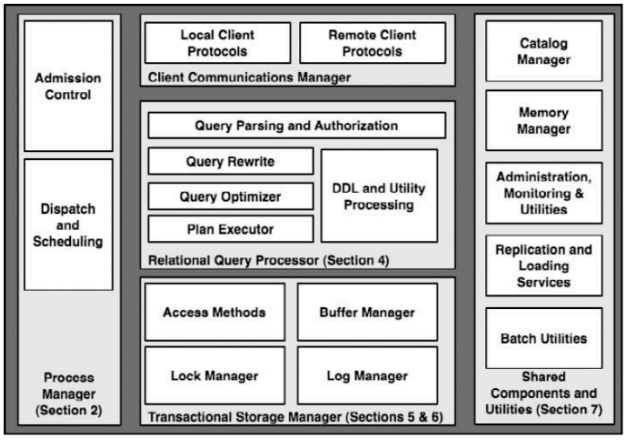


Figura 1.1: Arquitectura típica de un DBMS.

Capítulo 2

Descripción del sistema

2.1. Contexto

Para el desarrollo del proyecto se debe seguir el modelo arquitectural generalmente utilizado para un DBMS, como se muestra en la Figura 1. Sin embargo, note que debido a la simplificación y lo reducido del alcance del trabajo, hay muchos módulos que no van a existir o que no es necesario simular. En general un DBMS tiene los siguientes módulos:

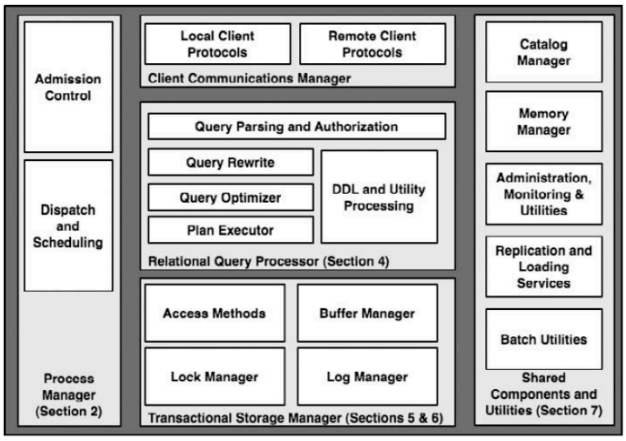


Figura 2.1: Arquitectura típica de un DBMS.

1. Módulo de Administración de Clientes y Administración de Procesos: El módulo de Administración de Clientes es el módulo encargado de recibir las conexiones de red de parte de los clientes del sistema. Este abre un socket entre el cliente y el DBMS y mantiene viva la conexión mientras la consulta es procesada y mientras la configuración tanto del

cliente como del DBMS no indiquen que hizo timeout. Este módulo se comunica con el módulo de Administración de procesos para asignar un hilo de ejecución a cada consulta.

2. Módulo de Procesamiento de Consultas: Este módulo es uno de los módulos principales de un DBMS relacional. En general se encarga de recibir sentencias escritas en SQL y de realizar:

   a) La validación léxica y sintáctica de la sentencia introducida.

   b) La validación semántica de la sentencia.

   c) La verificación de los permisos del usuario que realiza la sentencia.

   d) Optimización de la sentencia.

   e) La ejecución de los operadores de las consultas.

   Note que no vamos a implementar las verificaciones de seguridad para ver si un usuario puede o no ejecutar la consulta, por simplicidad vamos a asumir que si puede.

3. Módulo de Transacciones y Almacenamiento: Dado que el sistema es una simulación no vamos a tener los problemas relacionados con la ejecución concurrente de consultas, por lo que no vamos a implementar los algoritmos y protocolos para la ejecución correcta de transacciones concurrentes. De este módulo lo único que vamos a simular es la carga de los datos del repositorio persistente (que en un DBMS comercial es su sistema de archivos propietarios en disco).

2.2. Descripción del sistema a simular

El servidor de matrícula de la UCR utiliza una base de datos relacional implementada con PintoDB. Este sistema, que es utilizado por diversos sistemas de la UCR, en promedio recibe 30 conexiones por minuto, tiempo exponencial. Cada conexión envía al inicio una única sentencia SQL para ser ejecutada por el servidor y obtener los resultados. Las sentencias que recibe pueden ser de 4 tipos:

Tipo de Consulta	Probabilidad	Read Only
SELECT	30 %	Sí
UPDATE	25 %	No
JOIN	35 %	Sí
DDL	10 %	No

El Módulo de Administración de Clientes es capaz de manejar un número de k de conexiones simultáneas, donde el valor de k es un parámetro de configuración que define el administrador del servidor. Cuando el número de conexiones establecidas llega al límite, las conexiones nuevas simplemente son rechazadas. Pinto es un sistema simple, po rlo que losclientes deben realizar una conexión nueva cada vez que desean realizar una operación en la base de datos. Debido a esto, cuando una conexión ya devuelve el resultado esperado de la sentencia y los datos son enviados por la red, la conexión es cerrada. Además, el servidor tiene un parámetro de configuración para el tiempo máximo de espera en las conexiones, para evitar ataques de denegación de servicio. Este parámetro t, que es configurado por el administrador del sistema, hace que t segundos después de haber creado la conexión, el módulo de administración de conexiones cierre la conexión y termine el hilo de ejecución (asuma que la consulta termina su etapa actual si está en servicio, pero no pasa a la etapa siguiente). Una vez que se ha establecido la conexión, el módulo va a solicitarle al módulo de administración de procesos la creación de un nuevo hilo para la conexión. Como este llamado debe realizarse mediante un *system call* al sistema operativo, el tiempo que dura el módulo en crear el hilo y tenerlo listo para ejecución sigue una distribución normal con una media de 1 segundo y una varianza de 0.01 segundos cuadrados o sea es n ($\mu = 1, \sigma^2 = 0.02$). El módulo de administración de procesos puede realizar un *system call* a la vez, si hay más de una conexión, esta se pone en una cola de capacidad infinita. Una vez que el hilo está listo para ejecución, la consulta pasa al módulo de Procesamiento de Consultas, en donde cada etapa toma en promedio:

Etapa	Duración
Validación Léxica	1/10 segundo
Validación Sintáctica	distribución uniforme en el intervalo $0 <= x <= 1$ segundos
Validación Semántica	distribución uniforme en el intervalo $0 <= x <= 2$ segundos
Verificación de Permisos	0.7 segundos en promedio, distribución exponencial
Optimización de Consultas	0.1 segundos para sentencias read-only, 1/4 de segundo para sente

El módulo de procesamiento de consultas tiene n procesos disponibles para procesar consultas (donde n es un parámetro de la configuración del servidor), cuando estos están ocupados, las consultas son almacenadas en una cola, en el orden en que llegaron. Todas las etapas mencionadas anteriormente son realizadas de forma consecutiva por el procesador para cada consulta. Cuando el procesamiento de la consulta finalice, esta es pasada al módulo de transacciones y acceso a datos. Este módulo puede procesar `p` consultas a

la vez, donde p es un parámetro de la configuración del servidor, las demás consultas son enviadas a una cola, sin embargo, en esta etapa, la disciplina de la cola es por prioridades:

1. DDL
2. UPDATE
3. JOIN
4. SELECT

Cuando una sentencia DDL está en ejecución ninguna otra sentencia puede ejecutarse. En este caso, el módulo se espera a que terminen todas las transacciones en proceso, y luego ejecuta esta sentencia en uno de los procesos, con los otros p-1 procesos desocupados. El manejo de transacciones tiene un costo asociado a la cantidad máxima de elementos concurrentes que pueden ejecutarse, en este caso, el tiempo necesario para coordinar la ejecución concurrente depende del número de procesos concurrentes p que utilice el sistema. Cada transacción va a tomar un tiempo p * 0.03 segundos en coordinar la ejecución de una transacción. El mismo módulo es el encargado de obtener los datos de la base de datos para que se ejecute la sentencia correspondiente. En este caso, el número de bloques de disco que se cargan por cada consulta es de:

Sentencia	Bloques discos cargados
DDL	0
UPDATE	0
JOIN	distribución uniforme en 1, 2, 3, ..., 64
SELECT	1

Cada bloque de disco dura 1/10 de segundo en ser cargado de la base de datos. Una vez obtenidos los datos de disco, la sentencia vuelve al módulo de procesamiento de consultas, en donde se ejecuta, tomando un tiempo de B^2 milisegundos en ejecutar el algoritmo correspondiente, donde B es la cantidad de bloques que se cargaron de disco para la sentencia. En el caso de una sentencia UPDATE toma 1 segundo. El ejecutor de sentencias tiene una capacidad de ejecutar m sentencias en paralelo, donde m es un parámetro de la configuración del sistema. Este módulo maneja una cola de sentencias por ejecutar. Por último, cuando finaliza la ejecución, el resultado, el cuál en promedio es de B/64 bloques es devuelto al módulo de administración de conexiones, el cual toma el resultado y lo pone en la red, tomando un tiempo

de R segundos en transmitirse, donde R es el tamaño en bloques del resultado de la sentencia. Cuando termina de transmitirse los datos, la conexión cierra y se admite una conexión nueva.

Capítulo 3

Diagrama de clases

Diagrama de clases en la siguiente página

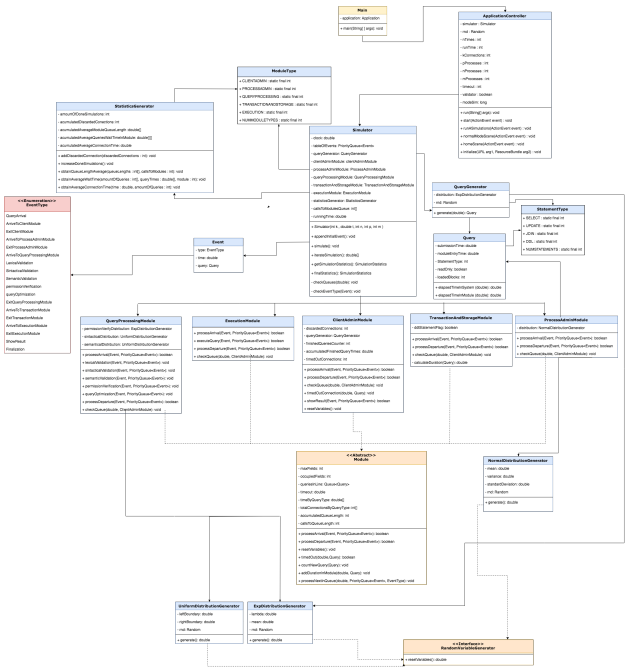
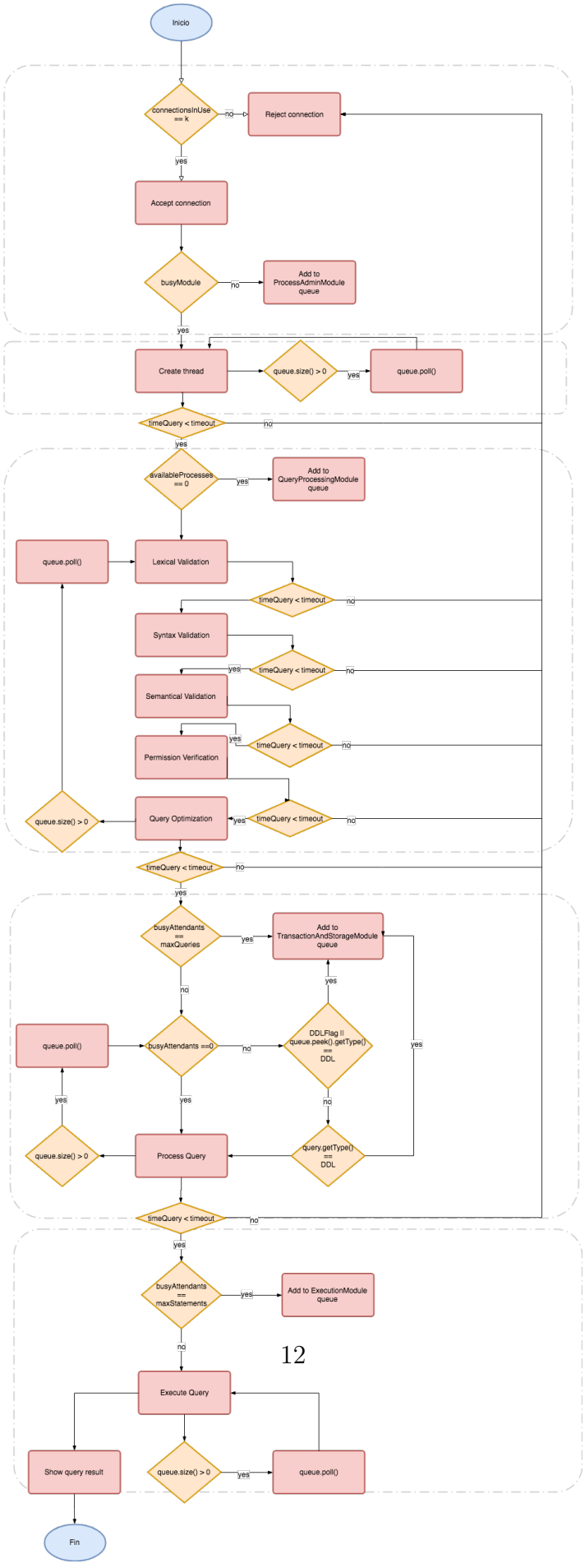


Figura 3.1: Diagrama de clases

Capítulo 4

Diagrama de flujo

Diagrama de Flujo en la siguiente página.


Inicio
connectionsInUse == k
no
Reject connection
yes
Accept connection
busyModule
no
Add to ProcessAdminModule queue
yes
Create thread
queue.size() > 0
yes
queue.poll()
timeQuery < timeout
no
yes
availableProcesses == 0
yes
Add to QueryProcessingModule queue
queue.poll()
Lexical Validation
timeQuery < timeout
no
Syntax Validation
yes
timeQuery < timeout
no
Semantical Validation
yes
timeQuery < timeout
no
Permission Verification
queue.size() > 0
Query Optimization
yes
timeQuery < timeout
no
timeQuery < timeout
no
yes
busyAttendants == maxQueries
yes
Add to TransactionAndStorageModule queue
no
yes
queue.poll()
busyAttendants ==0
no
DDLFlag || queue.peek().getType() == DDL
yes
yes
yes
no
queue.size() > 0
Process Query
query.getType() == DDL
timeQuery < timeout
no
yes
busyAttendants == maxStatements
yes
Add to ExecutionModule queue
no
Execute Query
Show query result
queue.size() > 0
yes
queue.poll()
Fin

Capítulo 5

Distribuciones utilizadas y su mecanismo de generación aleatoria

5.0.1. Valores Aleatorios de la distribución Normal

Para generar valores aleatorios que sigan una distribución normal utilizamos la fórmula:

$$Z = \sum_{i=0}^{12} R_i - 6$$

$$X = \mu + \sigma \cdot Z$$

Demostración:

Utilizamos el Algoritmo de Convolución que consiste en:

> El teorema del límite central o teorema central del límite indica que, en condiciones muy generales, si S_n es la suma de n variables aleatorias independientes y de varianza no nula pero finita, entonces la función de distribución de S_n «se aproxima bien» a una distribución normal con media n y varianza n 2.

$$z = \frac{\sum_{i=0}^{n} R_i - n \cdot 0.5}{(n/12)^{1/2}}$$

Y sabemos que utilizando 12 números aleatorios es suficiente para generar valores aleatorio normales, y entonces así queda la fórmula de arriba.

5.0.2. Valores Aleatorios de la distribución Exponencial

Para generar valores aleatorios que sigan una distribución exponencial utilizamos la fórmula:

$$\frac{-1}{\lambda} \cdot ln(r)$$

Demostración:
Sabemos que:

$$f(x) = \lambda \cdot e^{-\lambda \cdot x}$$

Utilizando el método de transformación inversa, entonces debemos calcular $F(x)$ **F grande**

$$F(x) = 1 - e^{-\lambda \cdot x}$$

Ahora igualamos $r = F(x)$

$$r = 1 - e^{-\lambda \cdot x}$$

Despejamos x:

$$r = 1 - e^{-\lambda \cdot x}$$
$$e^{-\lambda \cdot x} = 1 - r$$
$$-\lambda \cdot x = \ln(1 - r)$$
$$x = \frac{-1}{\lambda} \cdot \ln(1 - r)$$

Nota: El $1 - r$ puede ser cambiado por un r, pues es simplemente un número aleatorio, entonces da igual cual utilicemos, es mejor ahorrarse una operación de más.

Paso 1

Paso 2

5.0.3. Valores Aleatorios de la distribución Uniforme

Para generar valores aleatorios que sigan una distribución uniforme utilizamos la fórmula:

$$x = a + (b - a) \cdot r$$

Demostración:

Sabemos que:

$$f(x) = \frac{1}{(b-a)} \rightarrow Si\ a <= x <= b$$

$$f(x) = 0 \rightarrow En\ caso\ contrario$$

Utilizando el método de transformación inversa, entonces debemos calcular $F(x)$ **F grande**

$$F(x) = 0 \rightarrow Si\ x < a$$

$$F(x) = \frac{(x-a)}{(b-a)} \rightarrow Si\ a <= x <= b$$

$$F(x) = 1 \rightarrow Si\ x > b$$

Ahora igualamos $r = F(x)$

$$r = \frac{(x-a)}{(b-a)}$$

Despejamos x:

$$x = a + (b - a) \cdot r$$

Capítulo 6

Estadísticas obtenidas

Estadísticas obtenidas para los valores prueba:

1. Correr la simulación 15 veces
2. Con un tiempo de 15000 segundos
3. Con los siguientes parámetros:
   a) k = 15
   b) n = 3
   c) p = 2
   d) m = 1
   e) t = 15

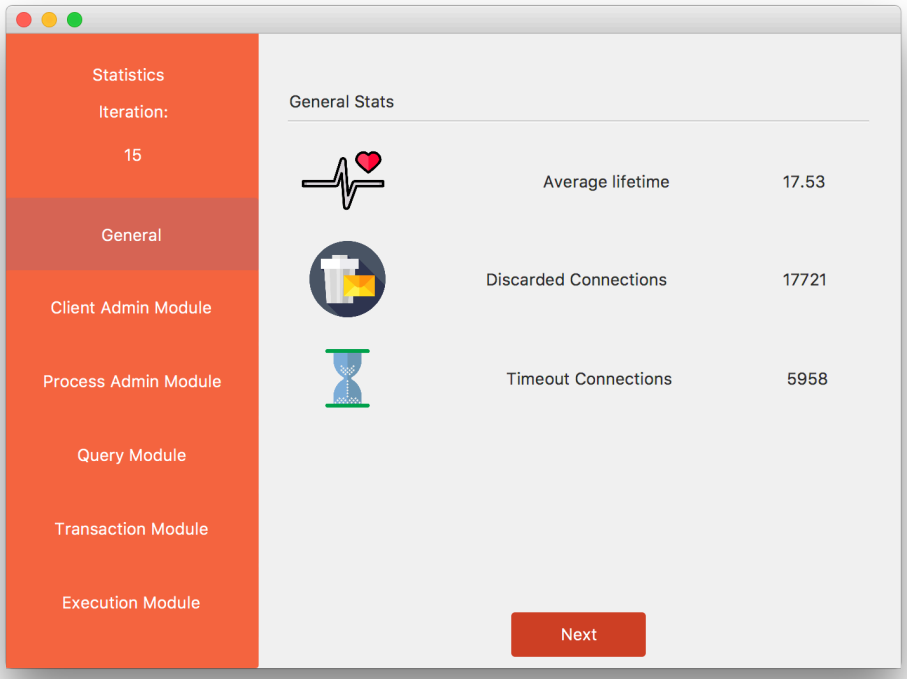


Figura 6.1: Iteración 15

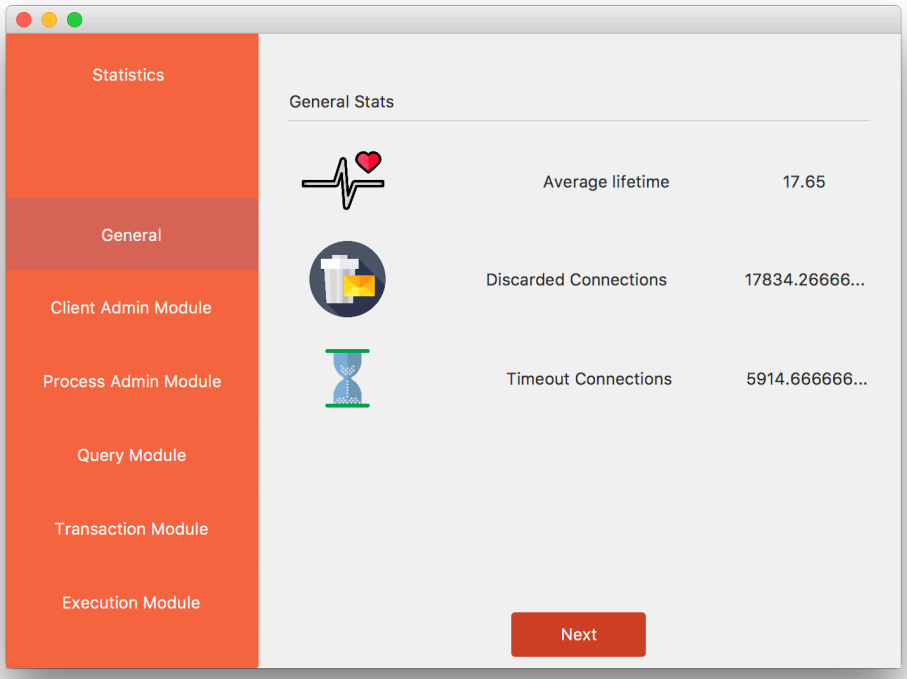


Figura 6.2: Estadísticas Generales

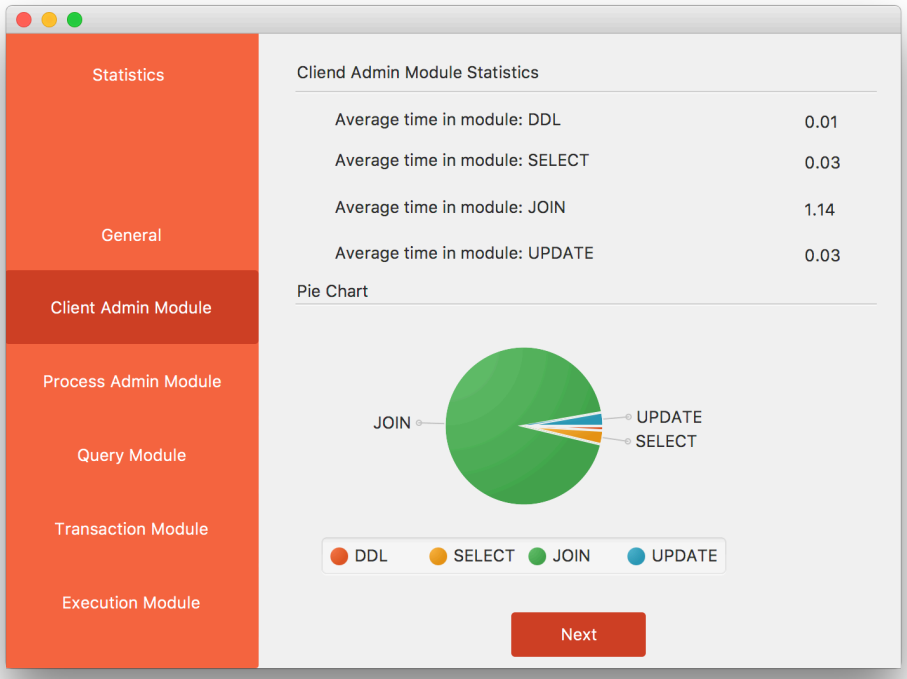


Figura 6.3: Estadísticas Client Admin Module

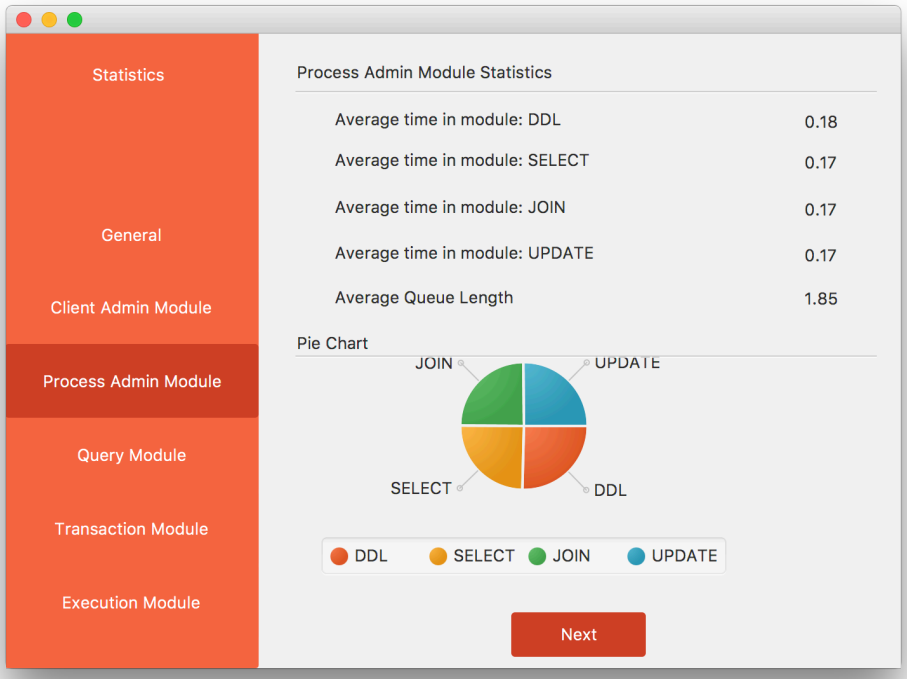


Figura 6.4: Estadísticas Process Admin Module

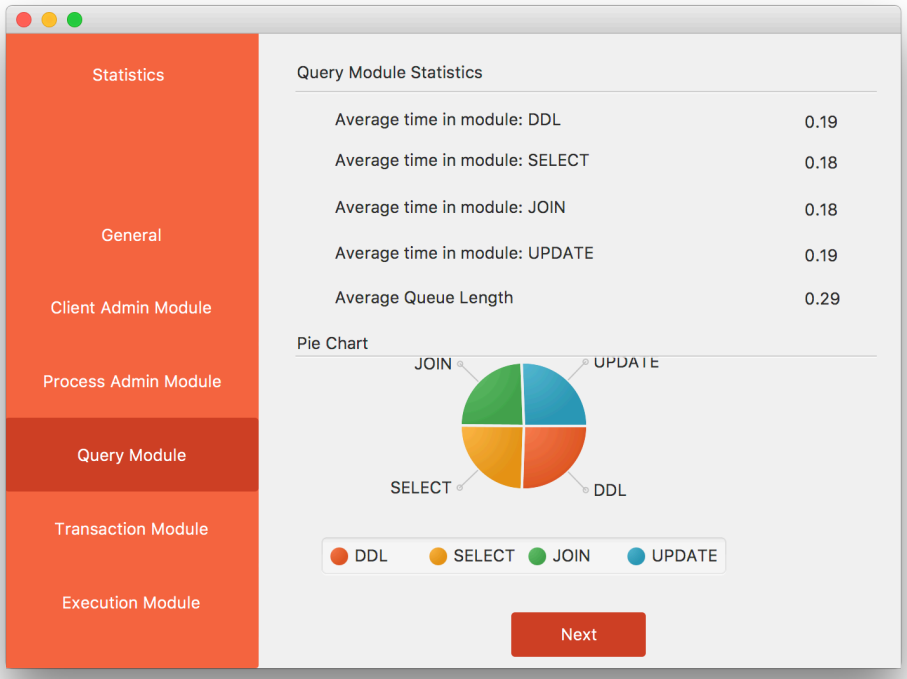


Figura 6.5: Estadísticas Query Module

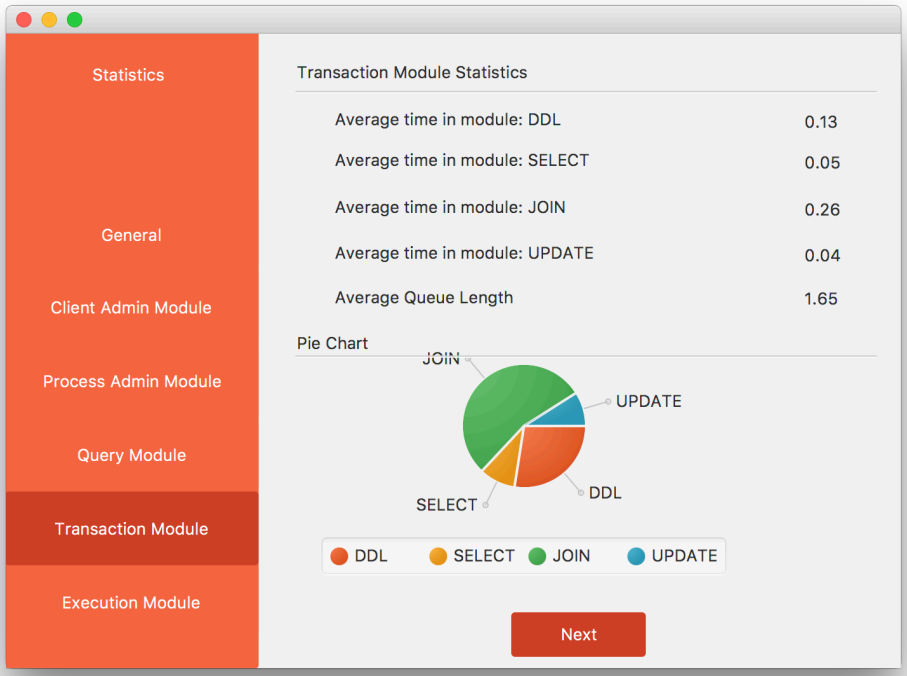


Figura 6.6: Estadísticas Transactions Module

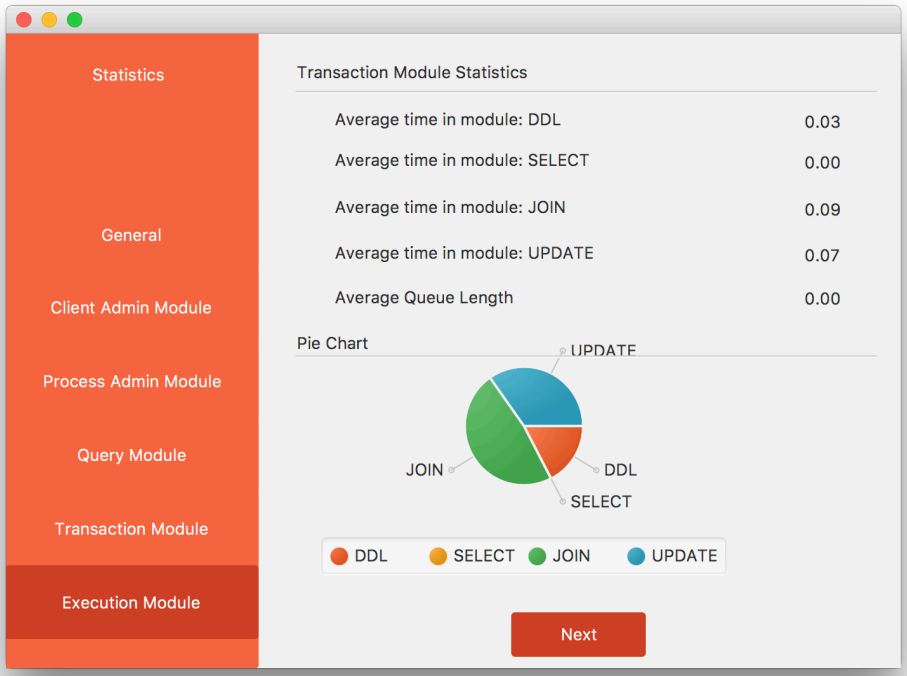


Figura 6.7: Estadísticas Execution Module

Capítulo 7

Análisis del sistema

1. ¿Qué pasa si los valores de n, p, k y m son 1 y t = 15 segundos?

   Con estos valores de la simulación sabemos varias cosas, primero en el sistema sólo va exister una consulta, por lo tanto no van a existir colas, además de que en la mayoría del tiempo los módulos van a estar ociosos mientras la única consulta no esté pasando por ellos.

2. ¿Qué parámetro influye más en el rendimiento del sistema?

3. ¿Qué cambios se podrían hacer para mejorar el rendimiento del problema?

4. Identifique la configuración adecuada para que la cantidad de conexiones descartadas sea menor al 5 %. Brinde el valor de los parámetros de configuración del sistema.

5. Como mejorar la eficiencia del sistema (Reduciendo el tiempo promedio de vida de una conexión)

Capítulo 8

Código fuente

8.1. Modules

8.1.1. Module

```
package DBMS_Sim.SourceCode;

import java.util.Iterator;
import java.util.PriorityQueue;
import java.util.Queue;

/**
 * This class consists in a parent class that contains some abstract methods a
 * our modules.
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */
public abstract class Module {
    //Maximum number of queries that we can execute at the same time
    protected int maxFields;
    //Number of queries we are attending
    protected int occupiedFields;
    protected Queue<Query> queriesInLine;
    //Maximum time of a connection in the system
    protected double timeout;
    //Statistical Variables
    protected double[] timeByQueryType;
```

```
protected int[] totalConnectionsByQueryType;
int acumulatedQueueLength;
int callsToQueueLength;


// ------------------------------------------------------------------------
// ----------------------------- Beginning of constructors section --------
// ------------------------------------------------------------------------

public Module(int maxFields, int occupiedFields, Queue<Query> queriesInLine,
    this.maxFields = maxFields;
    this.occupiedFields = occupiedFields;
    this.queriesInLine = queriesInLine;
    this.timeout = timeout;
    this.timeByQueryType = new double[StatementType.NUMSTATEMENTS];
    this.totalConnectionsByQueryType = new int[StatementType.NUMSTATEMENTS];
    this.acumulatedQueueLength = 0;
    this.callsToQueueLength = 0;
}

// ------------------------------------------------------------------------
// ------------------------------ End of the constructors section ---------
// ------------------------------------------------------------------------




// ------------------------------------------------------------------------
// ------------------------------- Beginning of methods section -----------
// ------------------------------------------------------------------------
public abstract boolean processArrival(Event event, PriorityQueue<Event> tab
public abstract boolean processDeparture(Event event, PriorityQueue<Event> t
public abstract void checkQueue(double clock, ClientAdminModule clientAdminM

public int getOccupiedFields(){return occupiedFields;}
public int getAcumulatedQueueLength() { return acumulatedQueueLength; }
public int getCallsToQueueLength() { return callsToQueueLength; }
public double[] getTimeByQueryType() { return timeByQueryType; }
public int[] getTotalConnectionsByQueryType() { return totalConnectionsByQue
```

```
/**
 * @param clock, current clock time.
 * @function checks if any query in the queue needs to be removed.
 */


/**
 * Asks for queueLength and adds it to an attribute for further statistics
 */
public void queueLength(){
    ++callsToQueueLength;
    acumulatedQueueLength += queriesInLine.size();
}

/**
 * Get the queue size
 */
public int queueSize(){
   return queriesInLine.size();
}

/**
 * @param clock, current clock time.
 * @param query, query that we wanna try to remove.
 * @return boolean that becomes true if the query was removed successfully
 * Checks if the query overstayed in the System, if so it removes it.
 */
protected boolean timedOut(double clock, Query query){
    boolean success = false;
    if(query.elapsedTimeInSystem(clock) >= this.timeout){
        success = true;
    }
    return success;
}

/**
 * @param query, the query that arrived to the Module.
 * Check the type of the query, and depending on which is it, increase the
 * that type of query.
 */
protected void countNewQuery(Query query){
```

```
    if(query.getStatementType() == StatementType.SELECT){
        ++totalConnectionsByQueryType[StatementType.SELECT];
    }else{
        if(query.getStatementType() == StatementType.UPDATE){
            ++totalConnectionsByQueryType[StatementType.UPDATE];
        }else{
            if(query.getStatementType() == StatementType.JOIN){
                ++totalConnectionsByQueryType[StatementType.JOIN];
            }else{
                if(query.getStatementType() == StatementType.DDL){
                    ++totalConnectionsByQueryType[StatementType.DDL];
                }else{
                    System.out.println("Unknown query type");
                }
            }
        }
    }
}

/**
 * @param clock, current clock time.
 * @param query, the query that is leaving the Module.
 * Check the type of the query, and depending on which is it, increase the
 * that type of query is staying in the Module.
 */
protected void addDurationInModule(double clock, Query query){
    double stayedTime = query.elapsedTimeInModule(clock);


    if(query.getStatementType() == StatementType.SELECT){
        timeByQueryType[StatementType.SELECT] += stayedTime;
    }else{
        if(query.getStatementType() == StatementType.UPDATE){
            timeByQueryType[StatementType.UPDATE] += stayedTime;
        }else{
            if(query.getStatementType() == StatementType.JOIN){
                timeByQueryType[StatementType.JOIN] += stayedTime;
            }else{
                if(query.getStatementType() == StatementType.DDL){
                    timeByQueryType[StatementType.DDL] += stayedTime;
```

```
                    }else{
                        System.out.println("Unknown query type");
                    }
                }
            }
        }

    protected void processNextInQueue(double clock, PriorityQueue<Event> tableOf
        Query nextQuery;
        Event newEvent;
        if(queriesInLine.size() > 0 && occupiedFields < maxFields){
            nextQuery = queriesInLine.poll();
            newEvent = new Event(eventType,clock, nextQuery);
            tableOfEvents.add(newEvent);
            queueLength();
        }else{
            --occupiedFields;
        }
    }

    /**
     * Resets the attributes of the module. Necessary if  new simulation wants
     */

    public void resetVariables(){
        this.occupiedFields= 0;
        queriesInLine.clear();

        for(int i = 0; i < StatementType.NUMSTATEMENTS; ++i){
            totalConnectionsByQueryType[i] = 0;
            timeByQueryType[i] = 0.0;
        }
    }

    // ---------------------------------------------------------------------
    // ------------------------------- End of the methods section ----------
    // ---------------------------------------------------------------------
}
```

8.1.2. Client Admin Module

```
package DBMS_Sim.SourceCode;

import java.util.PriorityQueue;

/**
 * This class simulates the Client Administration Module, simulates the arrival
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */


public class ClientAdminModule extends Module{
    private int discardedConnections;
    private QueryGenerator queryGenerator;
    private int finishedQueriesCounter;
    private double accumulatedFinishedQueryTimes;
    private int numberOfArrivalToTheSystem;
    private int timedOutConnections;

    // ---------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section -------
    // ---------------------------------------------------------------------

    public ClientAdminModule(int maxFields, double timeout){
        super(maxFields,0,new PriorityQueue<Query>(),timeout);
        queryGenerator = new QueryGenerator();
        finishedQueriesCounter = 0;
        accumulatedFinishedQueryTimes = 0;
        numberOfArrivalToTheSystem = 0;
        discardedConnections = 0;
    }

    // ---------------------------------------------------------------------
    // ------------------------------ End of the constructors section --------
    // ---------------------------------------------------------------------
```

```
// ------------------------------------------------------------------------
// ---------------------- Beginning of the setters and getters section ---
// ------------------------------------------------------------------------


public int getDiscardedConnections() {
    return discardedConnections;
}
public int getTimedOutConnections() {
    return timedOutConnections;
}

public QueryGenerator getQueryGenerator() { return queryGenerator;}
public int getFinishedQueriesCounter() { return finishedQueriesCounter; }
public double getAccumulatedFinishedQueryTimes() { return accumulatedFinishe

// ------------------------------------------------------------------------
// ------------------------- End of the setters and getters section -------
// ------------------------------------------------------------------------



// ------------------------------------------------------------------------
// ------------------------------ Beginning of methods section ------------
// ------------------------------------------------------------------------

/**
 * @param event, object that contains the information and the object needed
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed as a result of a timed
 * Creates an exit event from this module or adds a query to the queue, dep
 */
public boolean processArrival(Event event, PriorityQueue<Event> tableOfEvent

    if(occupiedFields < maxFields){
        //Generate a new arrival to next module

        event.setType(EventType.ArriveToProcessAdminModule);
        tableOfEvents.add(event);
        countNewQuery(event.getQuery());
```

```
            occupiedFields++;
        }else{
            discardedConnections++;
        }
        numberOfArrivalToTheSystem++;
        //We generate a new arrival to the system

        Query query = queryGenerator.generate(event.getTime());
        Event arrive = new Event(EventType.ArriveClientToModule,query.getSubmiss
        tableOfEvents.add(arrive);


        return false;
    }

    /**
     * @param event, object that contains the information and the object needed
     * @param tableOfEvents, queue with a list of events to be executed.
     * @return  boolean that says if a query was removed as a result of a time
     * Creates an arrival event for next module if query hasn't timed out and
     * If not, decrements occupiedFields. Calls another method for statistic pu
     */
    public boolean processDeparture(Event event, PriorityQueue<Event> tableOfEve
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());
        addDurationInModule(event.getTime(),event.getQuery());
        --occupiedFields;
        finishedQueriesCounter++;
        accumulatedFinishedQueryTimes += (event.getTime() - event.getQuery().get
        return false;
    }

    /**
     * @param clock, current simulation time.
     * @param query, query that timed out.
     * Modifies statistic variables after a connection timed out
     */
    public void timedOutConnection(double clock, Query query){

        --occupiedFields;
        ++timedOutConnections;
```

```
        //++discardedConnections;
        finishedQueriesCounter++;
        accumulatedFinishedQueryTimes += (clock - query.getSubmissionTime());

    }

    /**
     * @param event, event that hold the query which results will be shown.
     * @param tableOfEvents, queue with a list of events to be executed.
     * Shows results before leaving module.
     */
    public boolean showResult(Event event, PriorityQueue<Event> tableOfEvents) {
        boolean timedOut = timedOut(event.getTime(),event.getQuery());
        if(!timedOut){
            event.getQuery().setModuleEntryTime(event.getTime());
            event.setTime(event.getTime() +( event.getQuery().getLoadedBlocks())
            event.setType(EventType.ExitClientModule);
            tableOfEvents.add(event);
        }
        countNewQuery(event.getQuery());
        addDurationInModule(event.getTime(),event.getQuery());

        return timedOut;
    }

    @Override
    public void checkQueue(double clock, ClientAdminModule clientAdminModule){
    }

    @Override
    public void resetVariables(){
        super.resetVariables();
        finishedQueriesCounter = 0;
        accumulatedFinishedQueryTimes = 0;
        numberOfArrivalToTheSystem = 0;
        discardedConnections = 0;
        timedOutConnections = 0;
    }

    // -----------------------------------------------------------------------
    // ------------------------------ End of the methods section -------------
```

```
    // ----------------------------------------------------------------------
}
```

8.1.3. Process Admin Module

```
package DBMS_Sim.SourceCode;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.PriorityQueue;

/**
 * This class simulates the Process Administration Module, simulates the creat
 * it is done through system calls.
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */

public class ProcessAdminModule extends Module {
    private NormalDistributionGenerator distribution;

    // ----------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section -------
    // ----------------------------------------------------------------------

    public ProcessAdminModule(int maxFields, double timeout) {
        super(maxFields,0,new LinkedList<Query>(),timeout);
        distribution = new NormalDistributionGenerator(1,0.01);
    }

    // ----------------------------------------------------------------------
    // ------------------------------ End of the constructors section --------
    // ----------------------------------------------------------------------


    // ----------------------------------------------------------------------
    // ------------------------------ Beginning of methods section ----------
    // ----------------------------------------------------------------------
```

```java
/**
 * @param event, object that contains the information and the object needed
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed as a result of a timed
 * Creates an exit event from this module or adds a query to the queue, dep
 */
public boolean processArrival(Event event, PriorityQueue<Event> tableOfEvent
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut) {
        if (occupiedFields == 0) {
            occupiedFields++;
            event.setType(EventType.ExitProcessAdminModule);
            event.setTime(event.getTime() + distribution.generate());
            tableOfEvents.add(event);
        } else {
            queueLength();
            queriesInLine.add(event.getQuery());
        }
    }
    return timedOut;
}

/**
 * @param event, object that contains the information and the object needed
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed as a result of a timed
 * Creates an arrival event for next module if query hasn't timed out and
 * If not, decrements occupiedFields. Calls another method for statistic pu
 */
public boolean processDeparture(Event event, PriorityQueue<Event> tableOfEve
    boolean timedOut = timedOut(event.getTime(), event.getQuery());
    Query nextQuery;
    Event newEvent;
    if(queriesInLine.size() > 0){
        queueLength();
        nextQuery = queriesInLine.poll();
        newEvent = new Event(EventType.ExitProcessAdminModule,event.getTime(
        tableOfEvents.add(newEvent);
    }else{
        --occupiedFields;
```

```
        }
        if(!timedOut) {
            event.setType(EventType.ArriveToQueryProcessingModule);
            tableOfEvents.add(event);
        }

        //Statistics
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());


        return timedOut;
    }

    /**
     * @param clock, current clock time.
     * @param clientAdminModule, module where timeouts will be handled.
     * Checks if a query in queue has timed out, if so, removes this query fro
     */

    public void checkQueue(double clock, ClientAdminModule clientAdminModule){
        ArrayList<Query> queriesToRemove = new ArrayList<Query>();
        for(Query query : queriesInLine){
            if(timedOut (clock,query)){
                clientAdminModule.timedOutConnection(clock, query);
                queriesToRemove.add(query);
            }
        }

        for (Query query: queriesToRemove){
            queriesInLine.remove(query);
        }
    }

    // ------------------------------------------------------------------------
    // ------------------------------- End of the methods section -------------
    // ------------------------------------------------------------------------
}
```

8.1.4. Query Processing Module

```
package DBMS_Sim.SourceCode;

import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.PriorityQueue;

/**
 * This class simulates the Query Processing Module, where all kinds of valida
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */

public class QueryProcessingModule extends Module{
    private ExpDistributionGenerator permissionVerifyDistribution;
    private UniformDistributionGenerator semanticalDistribution;
    private UniformDistributionGenerator sintacticalDistribution;


    // ---------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section -------
    // ---------------------------------------------------------------------

    public QueryProcessingModule(int maxFields, double timeout){
        super(maxFields,0,new LinkedList<Query>(),timeout);
        setPermissionVerifyDistribution(new ExpDistributionGenerator(0.7));
        setSemanticalDistribution(new UniformDistributionGenerator(0,2));
        setSintacticalDistribution(new UniformDistributionGenerator(0,1));

    }

    // ---------------------------------------------------------------------
    // ------------------------------ End of the constructors section --------
    // ---------------------------------------------------------------------
```

```
// ------------------------------------------------------------------------
// ---------------------- Beginning of the setters and getters section ---
// ------------------------------------------------------------------------

public void setSintacticalDistribution(UniformDistributionGenerator sintacti
public void setSemanticalDistribution(UniformDistributionGenerator semantica
public void setPermissionVerifyDistribution(ExpDistributionGenerator permiss

public UniformDistributionGenerator getSintacticalDistribution() {
    return sintacticalDistribution;
}
public UniformDistributionGenerator getSemanticalDistribution() {
    return semanticalDistribution;
}
public ExpDistributionGenerator getPermissionVerifyDistribution() {
    return permissionVerifyDistribution;
}

// ------------------------------------------------------------------------
// ------------------------- End of the setters and getters section -------
// ------------------------------------------------------------------------



// ------------------------------------------------------------------------
// ------------------------------ Beginning of methods section ------------
// ------------------------------------------------------------------------

/**
 * @param event, object that contains the information and the object needed
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed as a result of a timed
 * Creates an exit event from this module or adds a query to the queue, dep
 */
public boolean processArrival(Event event, PriorityQueue<Event> tableOfEvent
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut) {
        if (occupiedFields < maxFields) {
            occupiedFields++;
```

```
            event.setType(EventType.LexicalValidation);
            event.setTime(event.getTime());
            tableOfEvents.add(event);
        } else {
            queueLength();
            queriesInLine.add(event.getQuery());
        }
    }
    return timedOut;
}


/**
 * @param event, object that contains the information needed to execute eac
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * If there is space validates the query that arrived and send it to the s
 */
public boolean lexicalValidation(Event event, PriorityQueue<Event> tableOfEv
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut) {
        event.setType(EventType.SintacticalValidation);
        event.setTime(event.getTime() + 0.1);
        tableOfEvents.add(event);

    }else{
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());
        processNextInQueue(event.getTime(),tableOfEvents,EventType.LexicalVa
    }

    return timedOut;
}

/**
 * @param event, object that contains the information needed to execute eac
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * Validates the query that arrived and send it to the semantic validation
 */
```

```
public boolean sintacticalValidation(Event event, PriorityQueue<Event> table
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut){
        event.setType(EventType.SemanticValidation);
        event.setTime(event.getTime() + sintacticalDistribution.generate());
        tableOfEvents.add(event);
    }else{
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());
        processNextInQueue(event.getTime(),tableOfEvents,EventType.LexicalVa
    }

    return timedOut;
}

/**
 * @param event, object that contains the information needed to execute ea
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * Validates the query that arrived and send it to be verified.
 */
public boolean semanticValidation(Event event, PriorityQueue<Event> tableOfE
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut){
        event.setType(EventType.PermissionVerification);
        event.setTime(event.getTime() + semanticalDistribution.generate());
        tableOfEvents.add(event);

    }else{
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());
        processNextInQueue(event.getTime(),tableOfEvents,EventType.LexicalVa
    }

    return timedOut;
}

/**
 * @param event, object that contains the information needed to execute ea
```

```
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * Verify the query that arrived and send it to be optimized.
 */
public boolean permissionVerification(Event event, PriorityQueue<Event> tabl
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut){
        event.setType(EventType.QueryOptimization);
        event.setTime(event.getTime() + permissionVerifyDistribution.generat
        tableOfEvents.add(event);

    }else{
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());
        processNextInQueue(event.getTime(),tableOfEvents,EventType.LexicalVa
    }

    return timedOut;
}

/**
 * @param event, object that contains the information needed to execute ea
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * Optimize the query nd send it to the transaction module.
 */
public boolean queryOptimization(Event event, PriorityQueue<Event> tableOfEv
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut){
        if(event.getQuery().getReadOnly()){
            event.setTime(event.getTime() + 0.1);
        }else{
            event.setTime(event.getTime() + 0.25);
        }
        event.setType(EventType.ExitQueryProcessingModule);
        tableOfEvents.add(event);
    }else{
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());
```

```
            processNextInQueue(event.getTime(),tableOfEvents,EventType.LexicalVa
        }
        return timedOut;
    }
    /**
     * @param event, object that contains the information needed to execute eac
     * @param tableOfEvents, queue with a list of events to be executed.
     * @return  boolean that says if a query was removed, so other modules can
     * Optimizes the query and sends it to the transaction module.
     */
    public boolean processDeparture(Event event, PriorityQueue<Event> tableOfEve
        boolean timedOut = timedOut(event.getTime(), event.getQuery());

        processNextInQueue(event.getTime(), tableOfEvents, EventType.LexicalVali

        if(!timedOut) {
            event.setType(EventType.ArriveToTransactionModule);
            tableOfEvents.add(event);
        }

        //Statistics
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());

        return timedOut;
    }
    /**
     * @param clock, object that contains the information needed to execute eac
     * @param clientAdminModule, queue with a list of events to be executed.
     * Checks if a query in queue need to be removed from queue due to a timeou
     */
    public void checkQueue(double clock, ClientAdminModule clientAdminModule){
        ArrayList<Query> queriesToRemove = new ArrayList<Query>();
        for(Query query : queriesInLine){
            if(timedOut(clock,query)){
                clientAdminModule.timedOutConnection(clock, query);
                queriesToRemove.add(query);
            }
        }

        for (Query query: queriesToRemove){
```

```
                queriesInLine.remove(query);
            }
        }

        // ------------------------------------------------------------------
        // ------------------------------ End of the methods section ------------
        // ------------------------------------------------------------------

}
```

8.1.5. Transaction and Storage Module

```
package DBMS_Sim.SourceCode;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

/**
 * This class simulates the Transaction and Storage Module, transaction execut
 * it's also in charge of loading data from disk.
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */


public class TransactionAndStorageModule extends Module{
    private boolean ddlStatementFlag;
    private UniformDistributionGenerator uniformDistributionGenerator;

    // ------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section --------
    // ------------------------------------------------------------------

    public TransactionAndStorageModule(int maxFields, double timeout){
        super(maxFields,0,new PriorityQueue<Query>(100, new Comparator<Query>()
            public int compare(Query arriving, Query queueHead) {
                int cmp = 0;
```

```
            if(arriving.getStatementType() > queueHead.getStatementType()){
                cmp = -1;
            }else{
                if(arriving.getStatementType() < queueHead.getStatementType(
                    cmp = 1;
                }
            }
            return cmp;
        }
    }),timeout);
    this.uniformDistributionGenerator = new UniformDistributionGenerator(1.0
}

// ------------------------------------------------------------------------
// ------------------------------ End of the constructors section ---------
// ------------------------------------------------------------------------



// ------------------------------------------------------------------------
// ----------------------- Beginning of the setters and getters section ---
// ------------------------------------------------------------------------

public void setDdlStatementFlag(boolean ddlStatementFlag) {
    this.ddlStatementFlag = ddlStatementFlag;
}

private boolean isDdlStatementFlag() {
    return ddlStatementFlag;
}

// ------------------------------------------------------------------------
// ------------------------- End of the setters and getters section -------
// ------------------------------------------------------------------------


// ------------------------------------------------------------------------
// ------------------------------ Beginning of methods section ------------
// ------------------------------------------------------------------------
```

```
/**
 * @param event, object that contains the information and the object needed
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed as a result of a timed
 * Creates an exit event from this module or adds a query to the queue, dep
 */
public boolean processArrival(Event event, PriorityQueue<Event> tableOfEvent
    boolean timedOut = timedOut(event.getTime(),event.getQuery());
    //We ask if the query that is arriving has timed out.
    if(!timedOut){
        event.getQuery().setModuleEntryTime(event.getTime());
        //We ask if there's enough space to serve the event.
        if(occupiedFields < maxFields) {
            //If no servers are occupied, proceed to be attended.
            if(occupiedFields == 0) {
                occupiedFields++;
                event.setType(EventType.ExitTransactionModule);
                event.setTime(event.getTime() + calculateDuration(event.getQ
                tableOfEvents.add(event);
            }else{

                //If there's occupied servers and the query arriving is a D
                if (event.getQuery().getStatementType() == StatementType.DDL
                    queueLength();
                    queriesInLine.add(event.getQuery());
                } else {
                    //If there is a DDL statement being processed, no query
                    if (isDdlStatementFlag()) {
                        queueLength();
                        queriesInLine.add(event.getQuery());
                    } else {
                        //If there's no DDL statement being processed, arri
                        occupiedFields++;
                        event.setType(EventType.ExitTransactionModule);
                        event.setTime(event.getTime() + calculateDuration(ev
                        tableOfEvents.add(event);
                    }
                }
            }
        }else{
            queueLength();
```

```
            queriesInLine.add(event.getQuery());
        }
    }

    return timedOut;
}

/**
 * @param event, object that contains the information and the object needed
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed as a result of a timed
 * Creates an arrival event for next module if query hasn't timed out and
 * If not, decrements occupiedFields. Calls another method for statistic pu
 */
public boolean processDeparture(Event event, PriorityQueue<Event> tableOfEve
    boolean timedOut = timedOut(event.getTime(),event.getQuery());
    Query nextQuery;
    Event newEvent;
    if(queriesInLine.size() > 0 && occupiedFields < maxFields){
        queueLength();
        nextQuery = queriesInLine.poll();
        newEvent = new Event(EventType.ExitTransactionModule,event.getTime()
        tableOfEvents.add(newEvent);
    }else{
        --occupiedFields;
    }
    if(!timedOut) {
        event.setType(EventType.ArriveToExecutionModule);
        tableOfEvents.add(event);
    }

    //Statistics
    addDurationInModule(event.getTime(),event.getQuery());
    countNewQuery(event.getQuery());

    return timedOut;
}

/**
 * @param clock, current clock time.
 * @param clientAdminModule, module where timeouts will be handled.
```

```
 * Checks if a query in queue has timed out, if so, removes this query fro
 */
@Override
public void checkQueue(double clock, ClientAdminModule clientAdminModule){
    ArrayList<Query> queriesToRemove = new ArrayList<Query>();
    for(Query query : queriesInLine){
        if(timedOut(clock,query)){
            clientAdminModule.timedOutConnection(clock, query);
            queriesToRemove.add(query);
        }
    }
    for (Query query: queriesToRemove){

        queriesInLine.remove(query);
    }
}

/**
 * @param beingProcessed, query that is currently being processed.
 * @return  The duration of the query being processed within this module.
 * Defines how long will this query last being attended and how many blocks
 */
private double calculateDuration(Query beingProcessed){
    double duration = maxFields * 0.03;
    int blocksLoaded = 1;

    switch(beingProcessed.getStatementType()){
        case StatementType.JOIN:
            blocksLoaded = (int)(uniformDistributionGenerator.generate() + 0
            duration += (0.1 * blocksLoaded);
            beingProcessed.setLoadedBlocks(blocksLoaded);
            break;
        case StatementType.SELECT:
            duration += 1/10;
            beingProcessed.setLoadedBlocks(blocksLoaded);
            break;
        default:
            break;
    }

    return duration;
```

```
    }

    // ------------------------------------------------------------------
    // ------------------------------ End of the methods section ---------
    // ------------------------------------------------------------------
}
```

8.1.6. Execution Module

```
package DBMS_Sim.SourceCode;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.PriorityQueue;

/**
 * This class simulates the Execution Module, this module executes all stateme
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */

public class ExecutionModule extends Module{

    // ------------------------------------------------------------------
    // ---------------------------- Beginning of constructors section -------
    // ------------------------------------------------------------------

    public ExecutionModule(int maxFields, double timeout){
        super(maxFields,0,new LinkedList<Query>(),timeout);
    }

    // ------------------------------------------------------------------
    // ---------------------------- End of the constructors section --------
    // ------------------------------------------------------------------



    // ------------------------------------------------------------------
    // ---------------------------- Beginning of methods section ----------
```

```
// ----------------------------------------------------------------------

/**
 * @param event, object that contains the information needed to execute ea
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * If there is space validates the query that arrived and send it to the e
 */

public boolean processArrival(Event event, PriorityQueue<Event> tableOfEvent
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut) {
        if (occupiedFields < maxFields) {
            occupiedFields++;
            event.setType(EventType.ExecuteQuery);
            event.setTime(event.getTime());
            tableOfEvents.add(event);
        } else {
            queriesInLine.add(event.getQuery());
        }
    }
    return timedOut;
}

/**
 * @param event, object that contains the information needed to execute ea
 * @param tableOfEvents, queue with a list of events to be executed.
 * @return  boolean that says if a query was removed, so other modules can
 * If there is space validates the query that arrived and send it to the e
 */
public boolean executeQuery(Event event, PriorityQueue<Event> tableOfEvents)
    boolean timedOut = timedOut(event.getTime(),event.getQuery());

    if(!timedOut) {
        event.setType(EventType.ExitExecutionModule);

        Query query = event.getQuery();
        double time = 0.0;
        if(query.getStatementType() == StatementType.DDL){
            time = 0.5;
```

```
            }else{
                if(query.getStatementType() == StatementType.UPDATE){
                    time = 1.0;
                }else{
                    time = query.getLoadedBlocks()*query.getLoadedBlocks()*0.001
                }
            }
            event.setTime(event.getTime() + time);

            tableOfEvents.add(event);

        }else{
            processNextInQueue(event.getTime(),tableOfEvents,EventType.ExecuteQu
            //Statistics
            addDurationInModule(event.getTime(),event.getQuery());
            countNewQuery(event.getQuery());
        }

        return timedOut;
    }

    /**
     * @param event, object that contains the information needed to execute ea
     * @param tableOfEvents, queue with a list of events to be executed.
     * @return  boolean that says if a query was removed, so other modules can
     * Send the query to the client admin module.
     */
    public boolean processDeparture(Event event, PriorityQueue<Event> tableOfEve
        boolean timedOut = timedOut(event.getTime(),event.getQuery());

        if(!timedOut){
            event.setType(EventType.ShowResult);
            tableOfEvents.add(event);
        }

        processNextInQueue(event.getTime(),tableOfEvents,EventType.ExecuteQuery)
        //Statistics
        addDurationInModule(event.getTime(),event.getQuery());
        countNewQuery(event.getQuery());

        return timedOut;
```

```
    }
    /**
     * @param clock, current clock time.
     * Checks if any query in the queue needs to be removed.
     */

    public void checkQueue(double clock, ClientAdminModule clientAdminModule){
        ArrayList<Query> queriesToRemove = new ArrayList<Query>();
        for(Query query : queriesInLine){
            if(timedOut (clock,query)){
                clientAdminModule.timedOutConnection(clock, query);
                queriesToRemove.add(query);
            }
        }

        for (Query query: queriesToRemove){
            queriesInLine.remove(query);
        }
    }
    // ---------------------------------------------------------------------
    // ------------------------------- End of the methods section ----------
    // ---------------------------------------------------------------------

}
```

8.2. Random Variable Generators

8.2.1. Random Variable Generator

```
package DBMS_Sim.SourceCode;
/**
 * This class is an interface that makes all distributions have a generate() me
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */
public interface RandomVariableGenerator{
    double generate();
}
```

8.2.2. Exponential Distribution Generator

```
package DBMS_Sim.SourceCode;

import java.util.Random;

public class ExpDistributionGenerator implements RandomVariableGenerator{
    private double mean;
    private double lambda;
    private Random rnd = new Random(System.currentTimeMillis());


    public ExpDistributionGenerator(double lambda){
        setLambda(lambda);
        setMean(1/lambda);
    }

    public void setLambda(double lambda){
        this.lambda = lambda;
    }

    public void setMean(double mean) {
        this.mean = mean;
    }

    public  double getLambda(){
        return lambda;
    }

    public double getMean() {
        return mean;
    }

    @Override
    public double generate() {
        double numRandom = rnd.nextDouble();
        return (-lambda)*Math.log(1-numRandom);
    }
}
```

8.2.3. Normal Distribution Generator

```
package DBMS_Sim.SourceCode;

import java.util.Random;

public class NormalDistributionGenerator implements RandomVariableGenerator{
    private double mean;
    private double variance;
    private double standardDeviation;
    private Random rnd = new Random(System.currentTimeMillis());


    public NormalDistributionGenerator(double mean, double variance){
        setMean(mean);
        setVariance(variance);
        setStandardDeviation(Math.sqrt(variance));
    }

    public void setMean(double mean){
        this.mean = mean;
    }

    public void setVariance(double variance){
        this.variance = variance;
    }

    public void setStandardDeviation(double standardDeviation) {
        this.standardDeviation = standardDeviation;
    }

    @Override
    public double generate() {
        double zeta= 0;
        double numRandom;

        for(int i = 0; i <12 ; ++i){
            numRandom = rnd.nextDouble();
            zeta += numRandom;
        }
        //System.out.println(zeta);
```

```
        zeta = zeta - 6;

        return  mean + standardDeviation * zeta;
    }

}
```

8.2.4. Uniform Distribution Generator

```
package DBMS_Sim.SourceCode;

import java.util.Random;

public class UniformDistributionGenerator implements RandomVariableGenerator{
    private double leftBoundary;
    private double rightBoundary;
    private Random rnd = new Random(System.currentTimeMillis());


    public UniformDistributionGenerator(double boundary1, double boundary2){
        if(boundary1 <= boundary2){
            setLeftBoundary(boundary1);
            setRightBoundary(boundary2);
        }else{
            setLeftBoundary(boundary2);
            setRightBoundary(boundary1);
        }
    }

    public void setLeftBoundary(double leftBoundary){
        this.leftBoundary = leftBoundary;
    }

    public void setRightBoundary(double rightBoundary){
        this.rightBoundary = rightBoundary;
    }

    public double getLeftBoundary(){
        return leftBoundary;
    }
```

```
    public double getRightBoundary(){
        return rightBoundary;
    }


    @Override
    public double generate() {
        double numRandom = rnd.nextDouble();
        //System.out.println(numRandom);
        return leftBoundary + (rightBoundary - leftBoundary) * numRandom;
    }

}
```

8.3. Query

```
package DBMS_Sim.SourceCode;

/**
 * This class holds relevant information for queries
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */
public class Query {
    private int loadedBlocks;
    private double moduleEntryTime;
    private boolean readOnly;
    private int statementType;
    private double submissionTime;



    // ------------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section --------
    // ------------------------------------------------------------------------

    public Query(boolean readOnly, int statementType, double submissionTime) {
        setLoadedBlocks(0);
```

```
    setReadOnly(readOnly);
    setSubmissionTime(submissionTime);
    setStatementType(statementType);
    setModuleEntryTime(-1);
}

// ------------------------------------------------------------------------
// ------------------------------ End of the constructors section ---------
// ------------------------------------------------------------------------



// ------------------------------------------------------------------------
// ----------------------- Beginning of the setters and getters section ---
// ------------------------------------------------------------------------


public void setLoadedBlocks(int loadedBlocks) { this.loadedBlocks = loadedBl
public void setModuleEntryTime(double moduleEntryTime) {
    this.moduleEntryTime = moduleEntryTime;
}
public void setReadOnly(boolean readOnly) {
    this.readOnly = readOnly;
}
public void setStatementType(int statementType) {
    this.statementType = statementType;
}
public void setSubmissionTime(double submissionTime) {
    this.submissionTime = submissionTime;
}

public int getLoadedBlocks() { return loadedBlocks; }
public double getModuleEntryTime() { return moduleEntryTime; }
public boolean getReadOnly() {
    return readOnly;
}
public double getSubmissionTime() {
    return submissionTime;
}
public int getStatementType() {
    return statementType;
```

```
    }

    public String toString(){
        String string = "Query's information: \n\t-Loaded blocks->" + loadedBloc
                "\n\t-Read only->" + readOnly + "\n\t-Submission time->" + submi
                statementType;
        return string;
    }


    // ------------------------------------------------------------------------
    // -------------------------- End of the setters and getters section ------
    // ------------------------------------------------------------------------



    // ------------------------------------------------------------------------
    // ------------------------------- Beginning of methods section -----------
    // ------------------------------------------------------------------------

    public double elapsedTimeInSystem(double clock){ return clock - submissionTi
    public double elapsedTimeInModule(double clock){
        return clock - moduleEntryTime;
    }

    // ------------------------------------------------------------------------
    // ------------------------------- End of the methods section -------------
    // ------------------------------------------------------------------------
}
```

8.4. Event

```
package DBMS_Sim.SourceCode;

/**
 * Instances of this class hold relevant information of an Event.
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
```

```
 */
public class Event {
    private EventType type;
    private double time;
    private Query query;


    // ------------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section --------
    // ------------------------------------------------------------------------

    public Event(EventType type, double time, Query query){
        setType(type);
        setTime(time);
        setQuery(query);
    }

    // ------------------------------------------------------------------------
    // ------------------------------ End of the constructors section ---------
    // ------------------------------------------------------------------------



    // ------------------------------------------------------------------------
    // ------------------------ Beginning of the setters and getters section ---
    // ------------------------------------------------------------------------

    public void setType(EventType type){
        this.type = type;
    }
    public void setTime(double time){
        this.time = time;
    }
    public void setQuery(Query query){
        this.query = query;
    }

    public EventType getType() {
        return type;
    }
    public double getTime() {
```

```
        return time;
    }
    public Query getQuery() {
        return query;
    }

    public String toString(){
        String string = "Event's information:\n\t-Type->" + type + "\n\t-Time->"
        return string;
    }

    // ------------------------------------------------------------------------
    // ------------------------- End of the setters and getters section -------
    // ------------------------------------------------------------------------
}
```

8.5. QueryGenerator

```
package DBMS_Sim.SourceCode;


import java.util.Random;

/**
 * Generate queries with the given distribution in the simulation description
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */
public class QueryGenerator {

    private ExpDistributionGenerator distribution;
    private Random rnd = new Random(System.currentTimeMillis());

    // ------------------------------------------------------------------------
    // ----------------------------- Beginning of constructors section --------
    // ------------------------------------------------------------------------

    public QueryGenerator(){
```

```
    //The media is 30 queries each minute, that means that in one second it
    //0.5 queries.
    distribution = new ExpDistributionGenerator(0.5);
}

// ------------------------------------------------------------------------
// ------------------------------ End of the constructors section ---------
// ------------------------------------------------------------------------



// ------------------------------------------------------------------------
// ----------------------- Beginning of the setters and getters section ---
// ------------------------------------------------------------------------

public void setDistribution(ExpDistributionGenerator distribution) {
    this.distribution = distribution;
}

public ExpDistributionGenerator getDistribution() {
    return distribution;
}

// ------------------------------------------------------------------------
// --------------------------- End of the setters and getters section -----
// ------------------------------------------------------------------------



// ------------------------------------------------------------------------
// ------------------------------ Beginning of methods section ------------
// ------------------------------------------------------------------------

/**
 * @param clock, current clock time.
 * @return the new query that is generated
 * Generates queries randomly.
 */
public Query generate(double clock){
    int newStatementType = StatementType.SELECT;
    boolean readOnly = true;
```

```
        double submissionTime = clock + distribution.generate();

        double numRandom = rnd.nextDouble();
        //System.out.println(numRandom);

        if(0.3 <= numRandom && numRandom <= 0.55){
            newStatementType = StatementType.UPDATE;
            readOnly = false;
        }else{
            if(0.55 < numRandom && numRandom <= 0.9){
                newStatementType = StatementType.JOIN;
            }else{
                if(numRandom > 0.9){
                    newStatementType = StatementType.DDL;
                    readOnly = false;
                }

            }
        }

        Query query = new Query(readOnly,newStatementType,submissionTime);
        return query;
    }

    // ------------------------------------------------------------------------
    // ------------------------------- End of the methods section -------------
    // ------------------------------------------------------------------------

}
```

8.6. EventType

```
package DBMS_Sim.SourceCode;

public enum EventType {
    ArriveClientToModule,
    ExitClientModule,
    ArriveToProcessAdminModule,
    ExitProcessAdminModule,
    ArriveToQueryProcessingModule,
```

```
    LexicalValidation,
    SintacticalValidation,
    SemanticValidation,
    PermissionVerification,
    QueryOptimization,
    ExitQueryProcessingModule,
    ArriveToTransactionModule,
    ExitTransactionModule,
    ArriveToExecutionModule,
    ExecuteQuery,
    ExitExecutionModule,
    ShowResult
}
```

8.7. ModuleType

```
package DBMS_Sim.SourceCode;

public class ModuleType {
    public final static int NUMMODULETYPES = 5;
    public final static int CLIENTADMIN = 0;
    public final static int PROCESSADMIN = 1;
    public final static int QUERYPROCESSING = 2;
    public final static int TRANSACTIONANDSTORAGE = 3;
    public final static int EXECUTION = 4;
}
```

8.8. StatementType

```
package DBMS_Sim.SourceCode;

public class StatementType {
    protected final static int NUMSTATEMENTS = 4;
    protected final static int SELECT = 0;
    protected final static int UPDATE = 1;
    protected final static int JOIN = 2;
    protected final static int DDL = 3;
}
```

8.9. Simulator

```
package DBMS_Sim.SourceCode;

import java.util.*;

/**
 * This class consists of the DBMS simulation body, it is where we track the po
 *
 * @author  Paulo Barrantes
 * @author  André Flasterstein
 * @author  Fabián Álvarez
 */
public class Simulator {

    private ClientAdminModule clientAdminModule;
    private double clock;
    private ExecutionModule executionModule;
    private ProcessAdminModule processAdminModule;
    private QueryGenerator queryGenerator;
    private QueryProcessingModule queryProcessingModule;
    private double runningTime;
    private StatisticsGenerator statisticsGenerator;
    private PriorityQueue<Event> tableOfEvents;
    private TransactionAndStorageModule transactionAndStorageModule;

    // ------------------------------------------------------------------------
    // ---------------------------- Beginning of constructors section --------
    // ------------------------------------------------------------------------
    public Simulator(int k, double t, int n, int p, int m){

        Comparator<Event> comparator = (event1, event2) -> {
            int cmp = 0;
            if (event1.getTime() < event2.getTime()) {
                cmp = -1;
            } else {
                if (event1.getTime() > event2.getTime()) {
                    cmp = 1;
                }
            }
            return cmp;
```

```
        };
        tableOfEvents = new PriorityQueue<>(100, comparator);
        clientAdminModule =  new ClientAdminModule(k, t);
        processAdminModule = new ProcessAdminModule(1,t);
        queryProcessingModule =  new QueryProcessingModule(n,t);
        transactionAndStorageModule = new TransactionAndStorageModule(p,t);
        executionModule = new ExecutionModule(m,t);
        queryGenerator = new QueryGenerator();
        statisticsGenerator = new StatisticsGenerator();
    }

    // ------------------------------------------------------------------------
    // ----------------------------- End of the constructors section ----------
    // ------------------------------------------------------------------------



    // ------------------------------------------------------------------------
    // ----------------------- Beginning of the setters and getters section ---
    // ------------------------------------------------------------------------
    public void setRunningTime(double runningTime) {
        this.runningTime = runningTime;
    }

    public double getRunningTime() {
        return runningTime;
    }

    public void setClock(double clock) {
        this.clock = clock;
    }

    public ClientAdminModule getClientAdminModule() {
        return clientAdminModule;
    }

    public double getClock() {
        return clock;
    }
    // ------------------------------------------------------------------------
    // ----------------------------- End of the constructors section ----------
```

```
// ------------------------------------------------------------------------


// ------------------------------------------------------------------------
// ------------------------------ Beginning of methods section ------------
// ------------------------------------------------------------------------



public void appendInitialEvent(){
    Query initialQuery = queryGenerator.generate(0);
    initialQuery.setSubmissionTime(0);
    Event initialArrive = new Event(EventType.ArriveClientToModule,0, initia
    tableOfEvents.add(initialArrive);
}

public void simulate(){
    appendInitialEvent();
    Event actualEvent;

    while(clock <= runningTime){
        actualEvent = tableOfEvents.poll();

        checkQueues(actualEvent.getTime());
        checkEventType(actualEvent);

        assert tableOfEvents.peek() != null;
        clock = tableOfEvents.peek().getTime();

    }
    getSimulationStatistics();

}


public double[] iterateSimulation(){
    double data []= new double[7];

    Event actualEvent =  tableOfEvents.poll();

    checkQueues(actualEvent.getTime());
```

```
        checkEventType(actualEvent);

        assert tableOfEvents.peek() != null;
        clock = tableOfEvents.peek().getTime();


        data[0] = clock;
        data[1] = (double) processAdminModule.queueSize();
        data[2] = (double) queryProcessingModule.queueSize();
        data[3] = (double) transactionAndStorageModule.queueSize();
        data[4] = (double) executionModule.queueSize();
        data[5] = (double) clientAdminModule.getDiscardedConnections();
        data[6] = (double) clientAdminModule.getTimedOutConnections();

        return data;
    }

    private void checkQueues(double clock){
        processAdminModule.checkQueue(clock,clientAdminModule);
        queryProcessingModule.checkQueue(clock,clientAdminModule);
        transactionAndStorageModule.checkQueue(clock, clientAdminModule);
        executionModule.checkQueue(clock, clientAdminModule);
    }

    private void checkEventType(Event actualEvent){
        boolean queryTimeOut;
        switch (actualEvent.getType()) {
            case ArriveClientToModule:
                clientAdminModule.processArrival(actualEvent, tableOfEvents);
                break;
            case ArriveToProcessAdminModule:
                queryTimeOut = processAdminModule.processArrival(actualEvent, ta
                if(queryTimeOut){
                    clientAdminModule.timedOutConnection(actualEvent.getTime(),
                }
                break;
            case ArriveToQueryProcessingModule:
                queryTimeOut =  queryProcessingModule.processArrival(actualEvent
                if(queryTimeOut){
                    clientAdminModule.timedOutConnection(actualEvent.getTime(),
                }
```

```
            break;
        case ArriveToTransactionModule:
            queryTimeOut = transactionAndStorageModule.processArrival(actual
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
            }
            break;
        case ArriveToExecutionModule:
            queryTimeOut = executionModule.processArrival(actualEvent, table
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
            }
            break;
        case ShowResult:
            queryTimeOut = clientAdminModule.showResult(actualEvent, tableOf
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
            }
            break;
        case ExitClientModule:
            queryTimeOut = clientAdminModule.processDeparture(actualEvent, t
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
            }
            break;
        case ExitProcessAdminModule:
            queryTimeOut= processAdminModule.processDeparture(actualEvent, t
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
            }
            break;
        case ExecuteQuery:
            queryTimeOut = executionModule.executeQuery(actualEvent, tableOf
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
            }
            break;
        case ExitTransactionModule:
            queryTimeOut = transactionAndStorageModule.processDeparture(actu
            if(queryTimeOut){
                clientAdminModule.timedOutConnection(actualEvent.getTime(),
```

```
        }
        break;
    case ExitExecutionModule:
        queryTimeOut = executionModule.processDeparture(actualEvent, tab
        if(queryTimeOut){
            clientAdminModule.timedOutConnection(actualEvent.getTime(),
        }
        break;
    case ExitQueryProcessingModule:
        queryTimeOut = queryProcessingModule.processDeparture(actualEven
        if(queryTimeOut){
            clientAdminModule.timedOutConnection(actualEvent.getTime(),
        }
        break;
    case LexicalValidation:
        queryTimeOut = queryProcessingModule.lexicalValidation(actualEve
        if(queryTimeOut){
            clientAdminModule.timedOutConnection(actualEvent.getTime(),
        }
        break;
    case SintacticalValidation:
        queryTimeOut = queryProcessingModule.sintacticalValidation(actua
        if(queryTimeOut){
            clientAdminModule.timedOutConnection(actualEvent.getTime(),
        }
        break;
    case SemanticValidation:
        queryTimeOut = queryProcessingModule.semanticValidation(actualEv
        if(queryTimeOut){
            clientAdminModule.timedOutConnection(actualEvent.getTime(),
        }

        break;
    case PermissionVerification:
        queryTimeOut =queryProcessingModule.permissionVerification(actua
        if(queryTimeOut){
            clientAdminModule.timedOutConnection(actualEvent.getTime(),
        }

        break;
    case QueryOptimization:
```

```
                queryTimeOut =queryProcessingModule.queryOptimization(actualEven
                if(queryTimeOut){
                    clientAdminModule.timedOutConnection(actualEvent.getTime(),
                }
                break;
            }
        }

        // ---------------------------------------------------------------------
        // ------------------------------ Statistical--------------------------
        // ---------------------------------------------------------------------



        public SimulationStatistics getSimulationStatistics(){

            SimulationStatistics simulationStatistics = new SimulationStatistics();

            simulationStatistics.setAcumulatedDiscardedConnections(clientAdminModule
            simulationStatistics.setTimeoutConnections(clientAdminModule.getTimedOut

            statisticsGenerator.addDiscardedConnections(simulationStatistics.getAcum
            statisticsGenerator.setTimeoutConnections(simulationStatistics.getTimeou
            double[] acumulatedModuleQueueLength = new double[ModuleType.NUMMODULETY
            acumulatedModuleQueueLength[ModuleType.CLIENTADMIN] = 0;

            acumulatedModuleQueueLength[ModuleType.PROCESSADMIN] = statisticsGenerat
            acumulatedModuleQueueLength[ModuleType.QUERYPROCESSING] = statisticsGene
            acumulatedModuleQueueLength[ModuleType.TRANSACTIONANDSTORAGE] = statisti
            acumulatedModuleQueueLength[ModuleType.EXECUTION] = statisticsGenerator.
            simulationStatistics.setAcumulatedModuleQueueLength(acumulatedModuleQueu

            double[][] acumulatedQueriesWaitTimeInModule = new double[ModuleType.NUM
            acumulatedQueriesWaitTimeInModule[ModuleType.CLIENTADMIN] = statisticsGe
            acumulatedQueriesWaitTimeInModule[ModuleType.PROCESSADMIN] = statisticsG
            acumulatedQueriesWaitTimeInModule[ModuleType.QUERYPROCESSING] = statisti
            acumulatedQueriesWaitTimeInModule[ModuleType.TRANSACTIONANDSTORAGE] = st
            acumulatedQueriesWaitTimeInModule[ModuleType.EXECUTION] = statisticsGene
            simulationStatistics.setAcumulatedQueriesWaitTimeInModule(acumulatedQuer

            simulationStatistics.setAcumulatedConnectionTime(statisticsGenerator.ave
```

```
            statisticsGenerator.increaseDoneSimulations();


            return simulationStatistics;
        }

        public SimulationStatistics finalStatistics(){
            SimulationStatistics simulationStatistics = new SimulationStatistics();
            simulationStatistics.setAcumulatedConnectionTime(statisticsGenerator.get
            simulationStatistics.setAcumulatedQueriesWaitTimeInModule(statisticsGene
            simulationStatistics.setAcumulatedModuleQueueLength(statisticsGenerator.
            simulationStatistics.setAcumulatedDiscardedConnections(statisticsGenerat

            return simulationStatistics;
        }

        public void reset() {
            tableOfEvents.clear();
            clock=0;
            runningTime = 0;
            clientAdminModule.resetVariables();
            processAdminModule.resetVariables();
            queryProcessingModule.resetVariables();
            transactionAndStorageModule.resetVariables();
            executionModule.resetVariables();
        }
        // ------------------------------------------------------------------------
        // ------------------------------- End of the methods section ------------
        // ------------------------------------------------------------------------

}
```

8.10. StatisticsGenerator

Capítulo 9

Problemas no resueltos

1. Tuvimos muchos problemas debuggeando
2. Tuvimos un problema faltando 5 minutos cuando estabamos compilando el sharelatex, lanzó un error.
3. No logramos realizar el análisis a fondo.
4. Creemos que tenemos errores.

Apéndice A

An Appendix may not be necessary

Text introducing the/this appendix.

A.1. Appendix section

Text of this section.
Subsections and further divisions can also be used in appendices.

Bibliografía