



Trabajo Práctico 2

15 / 11 / 2013

Bases De Datos

Grupo 4

Integrante	LU	Correo electrónico
Carreiro, Martin	45/10	martin301290@gmail.com
Kujawski, Kevin	459/10	kevinkuja@gmail.com
Ortiz De Zarate, Juan Manuel	403/10	jmanuoz@gmail.com
Teren, Leonardo	332/09	lteren@gmail.com



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

1. Introducción

El Buffer Manager es uno de los componentes más importantes dentro de un motor de BD. Su principal función es administrar un espacio de memoria de la BD, utilizado como una especie de memoria caché. El objetivo es que las diferentes aplicaciones que usan la BD y requieren páginas de disco, puedan recuperar la página de este espacio de memoria y accedan lo menos posible al disco. El espacio de memoria administrado por el Buffer Manager puede ser organizado de diferentes formas y la estrategia para decidir cuál página reemplazar cuando ya no queda más espacio también puede variar.

Este trabajo se encarga de implementar los algoritmos Touch Count, LRU y MRU, y compararemos su comportamiento a partir de posibles situaciones.

2. LRU

2.1. Descripción

El algoritmo Least Recently Used (LRU) descarta primero los elementos menos usados recientemente. El algoritmo lleva el seguimiento de lo que se va usando, lo que resulta caro si se quiere hacer con precisión. La implementación de esta técnica requiere llevar la cuenta de la edad de cada elemento de cache y buscar el menos usado en base a ella. En una implementación como esa, cada vez que que se usa un elemento, la edad de todos las demás elementos cambia.

2.2. Implementación

2.2.1. LRU Buffer Frame

```
package ubadb.core.components.bufferManager.bufferPool.replacementStrategies.lru;

import java.util.Date;

import ubadb.core.common.Page;
import ubadb.core.components.bufferManager.bufferPool.BufferFrame;
import ubadb.core.exceptions.BufferFrameException;

public class LRUBufferFrame extends BufferFrame
{
    private Date referencedDate;

    public LRUBufferFrame(Page page) {
        super(page);
        referencedDate = new Date();
    }

    public Date getReferencedDate()
    {
        return referencedDate;
    }

    public void pin(){
        super.pin();
        referencedDate = new Date();
    }

    public void unpin() throws BufferFrameException{
        super.unpin();
        referencedDate = new Date();
    }
}
```

2.2.2. LRU Buffer Frame Strategy

```
package ubadb.core.components.bufferManager.bufferPool.replacementStrategies.lru;

import java.util.Collection;
```

```

import java.util.Date;

import ubadb.core.common.Page;
import ubadb.core.components.bufferManager.bufferPool.BufferFrame;
import ubadb.core.components.bufferManager.bufferPool.
    replacementStrategies.PageReplacementStrategy;
import ubadb.core.exceptions.PageReplacementStrategyException;

public class LRURemplacementStrategy implements PageReplacementStrategy
{
    public BufferFrame findVictim(Collection<BufferFrame> bufferFrames) throws
        PageReplacementStrategyException
    {
        LRUBufferFrame victim = null;
        Date oldestReplaceablePageDate = null;

        for(BufferFrame bufferFrame : bufferFrames)
        {
            LRUBufferFrame lruBufferFrame = (LRUBufferFrame)
                bufferFrame; //safe cast as we know all frames are
                of this type
            if(lruBufferFrame.canBeReplaced() &&
                (oldestReplaceablePageDate==null ||
                 lruBufferFrame.getReferencedDate().
                     before(oldestReplaceablePageDate)))
            {
                victim = lruBufferFrame;
                oldestReplaceablePageDate =
                    lruBufferFrame.getReferencedDate();
            }
        }

        if(victim == null)
            throw new PageReplacementStrategyException("No page can
                be removed from pool");
        else
            return victim;
    }

    public BufferFrame createNewFrame(Page page)
    {
        return new LRUBufferFrame(page);
    }

    public String toString() {
        return "LRU Replacement Strategy";
    }
}

```

3. MRU

3.1. Descripción

MRU descarta primero -al contrario de LRU- los elementos más usados recientemente. Los algoritmos MRU son los más útiles en situaciones en las que cuanto más antiguo es un elemento, más probable es que se acceda a él.

3.2. Implementación

3.2.1. MRU Buffer Frame

```
package ubadb.core.components.bufferManager.bufferPool.replacementStrategies.mru;
import ubadb.core.common.Page;
import ubadb.core.components.bufferManager.bufferPool.BufferFrame;
import ubadb.core.exceptions.BufferFrameException;

import java.util.Date;

public class MRUBufferFrame extends BufferFrame
{
    private Date referencedDate;

    public MRUBufferFrame(Page page) {
        super(page);
        referencedDate = new Date();
    }

    public Date getReferencedDate()
    {
        return referencedDate;
    }

    public void pin(){
        super.pin();
        referencedDate = new Date();
    }

    public void unpin() throws BufferFrameException{
        super.unpin();
        referencedDate = new Date();
    }
}
```

3.2.2. MRU Buffer Frame Strategy

```
package ubadb.core.components.bufferManager.bufferPool.replacementStrategies.mru;

import java.util.Collection;
import java.util.Date;

import ubadb.core.common.Page;
import ubadb.core.components.bufferManager.bufferPool.BufferFrame;
import ubadb.core.components.bufferManager.bufferPool.
    replacementStrategies.PageReplacementStrategy;
import ubadb.core.exceptions.PageReplacementStrategyException;

public class MRUReplacementStrategy implements PageReplacementStrategy
{
    public BufferFrame findVictim(Collection<BufferFrame> bufferFrames) throws
        PageReplacementStrategyException
    {
        MRUBufferFrame victim = null;
        Date newestReplaceablePageDate = null;
    }
}
```

```

    for(BufferFrame bufferFrame : bufferFrames)
    {
        MRUBufferFrame mruBufferFrame = (MRUBufferFrame)
            bufferFrame; //safe cast as we know all frames are
                        //of this type
        if(mruBufferFrame.canBeReplaced() &&
            (newestReplaceablePageDate==null
             || mruBufferFrame.getReferencedDate().
                 after(newestReplaceablePageDate)))
        {
            victim = mruBufferFrame;
            newestReplaceablePageDate =
                mruBufferFrame.getReferencedDate();
        }
    }

    if(victim == null)
        throw new PageReplacementStrategyException("No page can
            be removed from pool");
    else
        return victim;
}

public BufferFrame createNewFrame(Page page)
{
    return new MRUBufferFrame(page);
}

public String toString() {
    return "MRU Replacement Strategy";
}
}

```

4. Touch Count

4.1. Descripción

El algoritmo que utiliza Oracle para manejar las páginas del Buffer Pool es conocido como “Touch Count” y es una variante del popular LRU.

4.1.1. Hot N Cold

Que significa
Como se mantiene balanceado
Como se crea

4.1.2. Incremento Touch Count

Lo de los 3 segundos
pin y unpin

4.1.3. Hot N Cold Movement

Si es mayor a 2
Mantener balanceo

4.2. Implementación

4.2.1. Touch Count Buffer Frame

```
package
    ubadb.core.components.bufferManager.bufferPool.replacementStrategies.touchcount;

import java.util.Date;

import ubadb.core.common.Page;
import ubadb.core.components.bufferManager.bufferPool.BufferFrame;
import ubadb.core.exceptions.BufferFrameException;

public class TouchBufferFrame extends BufferFrame implements
    Comparable<TouchBufferFrame>{

        public Integer count;
        public Date lastTouch;

        public TouchBufferFrame(Page page) {
            super(page);
            count = 0;
            lastTouch = new Date();
        }

        public void pin(){
            super.pin();
            increaseCount();
        }

        public void unpin() throws BufferFrameException{
            super.unpin();
            increaseCount();
        }

        public void increaseCount(){
            Date now = new Date();
```

```

        @SuppressWarnings("unused")
        long difference = (long) ((now.getTime() -
            lastTouch.getTime())/1000);

        if((now.getTime() - lastTouch.getTime())/1000 >= 3){
            count++;
            lastTouch = new Date();
        }
    }

    @Override
    public int compareTo(TouchBufferFrame arg0) {
        return count.compareTo(((TouchBufferFrame)arg0).count);
    }
}

```

4.2.2. Touch Count Buffer Frame Strategy

```

package
    ubadb.core.components.bufferManager.bufferPool.replacementStrategies.touchcount;

import java.util.Collection;
import java.util.LinkedList;

import ubadb.core.common.Page;
import ubadb.core.components.bufferManager.bufferPool.BufferFrame;
import ubadb.core.components.bufferManager.bufferPool.
    replacementStrategies.PageReplacementStrategy;
import ubadb.core.exceptions.PageReplacementStrategyException;

public class TouchCountReplacementStrategy implements PageReplacementStrategy {

    private LinkedList<TouchBufferFrame> cold;
    private LinkedList<TouchBufferFrame> hot;

    public BufferFrame findVictim(Collection<BufferFrame> bufferFrames)
        throws PageReplacementStrategyException {

        hotNColdMovement();

        TouchBufferFrame victim = firstColdFrame();
        return victim;
    }

    private TouchBufferFrame firstColdFrame() throws
        PageReplacementStrategyException{
        for (TouchBufferFrame bufferFrame : cold) {
            if (bufferFrame.canBeReplaced()){
                cold.remove(bufferFrame);

                if(Math.abs(cold.size() - hot.size())>0){
                    cold.addLast(hot.removeLast());
                }

                return bufferFrame;
            }
        }
        throw new PageReplacementStrategyException("No hay Cold Buffer
            reemplazable");
    }

    private void hotNColdMovement(){
        for(TouchBufferFrame bufferFrame : cold){

```

```

        if(bufferFrame.count > 2 ){
            bufferFrame.count = 0;
            hot.addFirst(bufferFrame);
            cold.remove(bufferFrame);
            cold.addLast(hot.removeLast());
        }
    }

}

public BufferFrame createNewFrame(Page page) {
    TouchBufferFrame bufferFrame = new TouchBufferFrame(page);

    cold.addFirst(bufferFrame);

    if (cold.size() >= 2){
        TouchBufferFrame coldToHodFrame = cold.pop();
        hot.addLast(coldToHodFrame);
    }

    return bufferFrame;
}

public String toString() {
    return "Touch Count Replacement Strategy";
}

}

```


5. Comparaciones

6. Conclusión