# Memòria

# Anàlisi i comparativa de frameworks dins de les 3 capes (Vista-Model-Dades) d'una arquitectura típica JEE

# Capa de dades

**Autor:** Bernat Requesens Fernández
**Tutor:** Javier Ferró Garcia

# ÍNDEX DEL DOCUMENT

# 1. Introducció

## *1.1. Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC*

En el context de J2EE hi ha un nombre elevat de tecnologies relacionades, i sovint actuen conjuntament per tal de complir amb diferents funcionalitats que demanda el mercat.

Dins d'aquestes tecnologies es poden distribuir en les diferents funcionalitats a les que donen suport: presentació de pantalles, accés a dades, integració de sistemes entre d'altres; en el cas d'aquest PFC es centra en valorar les que corresponen a les capes més usuals dins de les aplicacions basades en J2EE.

- **Presentació**: El que veu l'usuari (per exemple la pàgina web que interpreta el navegador).

- **Negoci**: Permet desenvolupar la funcionalitat del producte. En aquesta capa es defineixen els punts de control de la funcionalitat (per exemple es valida si l'estat en el que ens trobem ens permet fer una acció o no).

- **Persistència**: Accedeix als diferents sistemes de dades (per exemple a un Sistema Gestor de Bases de Dades), però també es pot utilitzar aquest concepte per a accedir a altres tipus d'emmagatzematge de dades.

Amb el temps han anat sorgint diferents solucions per a les tres capes anomenades anteriorment, per tal de facilitar i evitar errors de codificació.

Aquestes solucions es distribueixen sota el nom de Framework i ofereixen una capa d'abstracció al programador per a realitzar operacions. Un Framework ha de permetre augmentar el temps de presa de requeriments en contrapartida del temps de desenvolupament.

## 1.2. Objectius Generals del PFC

Entre els objectius principals del projecte hi consten les següents fites: donat un conjunt de Framework d'una de les capes, mesurar:

1) Mitjançant proves de codi que realitzen la mateixa tasca utilitzant els diferents Framework escollits la seva eficiència (velocitat i memòria entre d'altres).

2) Paràmetres que tenen a veure amb la facilitat de codificació utilitzant els diferents Framework (nombre de línies, claredat del codi, nombre de fitxers de configuració, corba d'aprenentatge, casos pràctics, projectes que treballen amb el Framework, suport a incidències...).

3) Compliment de la pròpia especificació del Framework (es valida que el Framework fa el que realment diu que fa).

## 1.3. Planificació del projecte

Tot i que la valoració del projecte es mostra en dies, cal tenir en compte com s'ha fet el càlcul d'aquests dies:

Relació crèdit / hores

| Crèdit | Hores |
|---|---|
| 1 | 15 |

PFC (total 9 crèdits)

| Crèdits | Hores |
|---|---|
| 9 | 135 |

Relació hores / PFC

| Setmana | Setmana nº | Hores setmana | Hores acumulades |
|---|---|---|---|
| 25/02/2008 | 1 | 7,5 | 7,5 |
| 03/03/2008 | 2 | 7,5 | 15 |
| 10/03/2008 | 3 | 7,5 | 22,5 |
| 17/03/2008 | 4 | 7,5 | 30 |
| 24/03/2008 | 5 | 7,5 | 37,5 |
| 31/03/2008 | 6 | 7,5 | 45 |
| 07/04/2008 | 7 | 7,5 | 52,5 |
| 14/04/2008 | 8 | 7,5 | 60 |
| 21/04/2008 | 9 | 7,5 | 67,5 |
| 28/04/2008 | 10 | 7,5 | 75 |
| 05/05/2008 | 11 | 7,5 | 82,5 |
| 12/05/2008 | 12 | 7,5 | 90 |
| 19/05/2008 | 13 | 7,5 | 97,5 |
| 26/05/2008 | 14 | 7,5 | 105 |
| 02/06/2008 | 15 | 7,5 | 112,5 |
| 09/06/2008 | 16 | 7,5 | 120 |
| 16/06/2008 | 17 | 7,5 | 127,5 |
| 23/06/2008 | 18 | 7,5 | 135 |

Hores de dedicació diària per a establir una base de temps

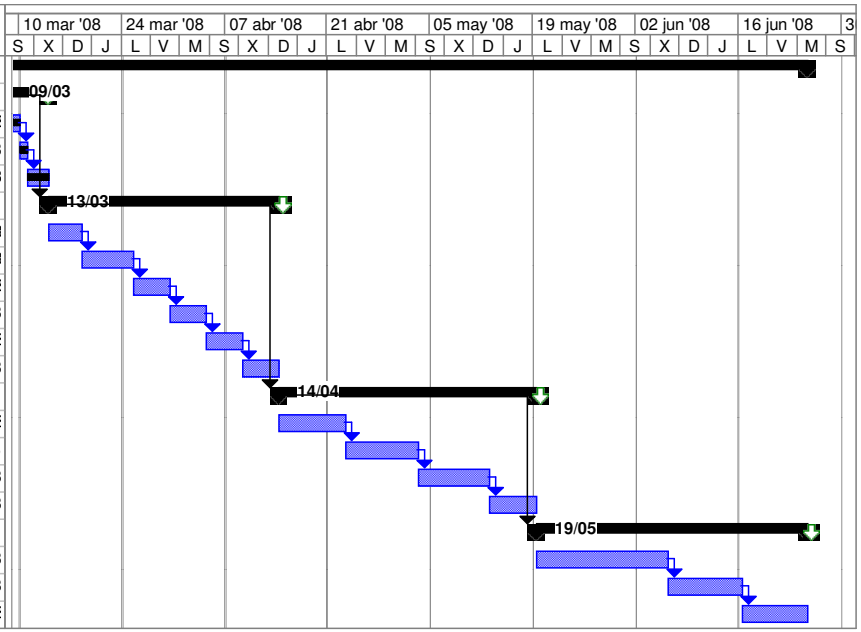
| Hores / Dia |
|---|
| 1h:05min |

### *1.3.1. Tasques a realitzar*

#### 1.3.1.1. Planificació de les tasques

Les tasques es divideixen entre les diferents entregues a realitzar del PFC:

| Id | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **PFC UOC** | **110 días** | **dom 09/03/08** | **mié 25/06/08** |
| 2 | **PAC1** | **5 días** | **dom 09/03/08** | **jue 13/03/08** |
| 3 | PAC 1 > Introducció | 1 día | dom 09/03/08 | dom 09/03/08 |
| 4 | PAC 1 > Definició objectius | 1 día | dom 09/03/08 | lun 10/03/08 |
| 5 | PAC 1 > Planificació de projecte | 3 días | lun 10/03/08 | jue 13/03/08 |
| 6 | **PAC2** | **32 días** | **jue 13/03/08** | **lun 14/04/08** |
| 7 | PAC2 > Definir capa estudi | 5 días | jue 13/03/08 | mar 18/03/08 |
| 8 | PAC2 > Seleccionar Frameworks a avaluar | 7 días | mar 18/03/08 | mar 25/03/08 |
| 9 | PAC2 > Definir proves automatitzades de mesura | 5 días | mar 25/03/08 | dom 30/03/08 |
| 10 | PAC2 > Definir mesures no automatitzades | 5 días | dom 30/03/08 | vie 04/04/08 |
| 11 | PAC2 > Definir mesures de compliment d'especificació del propi Framework | 5 días | vie 04/04/08 | mié 09/04/08 |
| 12 | PAC2 > Definir el entorn de treball a utilitzar | 5 días | mié 09/04/08 | lun 14/04/08 |
| 13 | **PAC3** | **36 días** | **lun 14/04/08** | **lun 19/05/08** |
| 14 | PAC3 > Configuració entorn de treball | 9 días | lun 14/04/08 | mié 23/04/08 |
| 15 | PAC3 > Codificació de les proves automatitzades | 10 días | mié 23/04/08 | sáb 03/05/08 |
| 16 | PAC3 > Executar mesures no automatitzades | 10 días | sáb 03/05/08 | lun 12/05/08 |
| 17 | PAC3 > Executar mesures de compliment d'especficació del propi Framework | 7 días | lun 12/05/08 | lun 19/05/08 |
| 18 | **Entrega Final** | **37 días** | **lun 19/05/08** | **mié 25/06/08** |
| 19 | Entrega final > Revissió i finalització del codi alpha | 18 días | lun 19/05/08 | vie 06/06/08 |
| 20 | Entrega final > Finalitzar documentació | 10 días | vie 06/06/08 | lun 16/06/08 |
| 21 | Entrega final > realització de presentació virtual | 9 días | lun 16/06/08 | mié 25/06/08 |



#### 1.3.1.2. Descripció de les tasques a realitzar

Descripció i temps d'execució de les tasques:

PAC1

| Tasca | Descripció | Temps |
|---|---|---|
| PAC 1 > Introducció | Definir introducció, cerca informació i redacció | 1 dia |
| PAC 1 > Definició d'objectius | Definició i redacció dels objectius del PFC | 1 dia |
| PAC 1 > Planificació de projecte | Detallar tasques a realitzar, realització de càlcul de temps basat en esforç. | 3 dies |

PAC2

| Tasca | Descripció | Temps |
|---|---|---|
| PAC2 > Definir de la capa d'estudi | Decidir la capa d'estudi | 5 dies |
| PAC2 > Seleccionar Frameworks a avaluar | Seleccionar Framewoks a avaluar | 7 dies |
| PAC2 > Definir proves automatitzades de mesura | Definir proves a realitzar per a obtenir mesures a partir d'execució de codi | 5 dies |
| PAC2 > Definir mesures no automatitzades | Definir paràmetres de mesura a partir de: la documentació, suport, corba d'aprenentatge, configuració i facilitat de codificació (per exemple: nombre de línies) | 5 dies |
| PAC2 > Definir mesures de compliment d'especificació del propi Framework | Recuperar especificació i determinar quins punts de control es realitzen per a comprovar el funcionament | 5 dies |
| PAC2 > Definir l'entorn de treball a utilitzar | Determinar, a partir de les necessitats de les proves a realitzar, un entorn de treball | 5 dies |

PAC3

| Tasca | Descripció | Temps |
|---|---|---|
| PAC3 > Configuració entorn de treball | Configuració de l'entorn, d'acord a l'especificació de la PAC 2 | 9 dies |
| PAC3 > Codificació de les proves automatitzades | Codificació de proves amb els Frameworks versió alpha | 10 dies |
| PAC3 > Executar mesures no automatitzades | Realitzar i avaluar mesures d'acord a l'especificació de la PAC2 | 10 dies |
| PAC3 > Executar mesures de compliment d'especificació del propi Framework | Realitzar i avaluar mesures d'acord a l'especificació de la PAC2 | 7 dies |

Entrega final

| Tasca | Descripció | Temps |
|---|---|---|
| Entrega final > Revisió i finalització del codi alpha | Revisió i modificació del codi alpha per a versió 1.0 | 18 dies |
| Entrega final > Finalitzar documentació | Redacció i revisió de documentació | 10 dies |
| Entrega final > Realització de presentació virtual | Realització de presentació virtual (20 diapositives) | 9 dies |

# 2. Anàlisi i definició del projecte

Durant aquest capítol es parla de les diferents decisions que es prenen abans de començar a realitzar codi (fase implementació). Concretament en aquest projecte es defineix:

- Capa d'estudi
- FW subjectes a estudi
- Proves a realitzar a cadascun dels FW
- Definició de l'entorn de treball per a la fase d'implementació

## *2.1 Definir la capa d'estudi a estudiar*

Seguint el model de tres capes (molt generalitzat dins del món de les aplicacions basades en J2EE), s'analitza la possibilitat de realitzar l'estudi sobre una d'aquestes tres capes: Presentació, Negoci i Persistència. Durant aquest punt es vol determinar quina capa serà l'objecte d'estudi del PFC.

### 2.1.2 Presentació

La capa de presentació és la que al final visualitza l'usuari (si el projecte és un pàgina web, aleshores serà el propi codi que es genera a la pàgina HTML). En aquest camp hi ha diferents FW que es poden utilitzar per tal de facilitar la codificació al programador.

Els FW de presentació, estan basats en **taglibs**, és a dir, en llibreries de **tags** que es poden utilitzar per a que la pròpia llibreria transformi el **tag** en codi HTML que el navegador podrà interpretar.

En aquest camp hi ha dos FW que s'utilitzen habitualment en els entorns web: Spring y Struts.

Tot i no ser únicament FW de presentació incorporen moltes llibreries per tal que la presentació sigui més senzilla.

### 2.1.3 Negoci

La capa de negoci és en la que s'avaluen les diferents condicions d'entrada (per exemple entrades d'un formulari) i s'executen les accions necessàries per a aconseguir l'efecte desitjat per l'usuari (desar dades, enviar correu, etc.). En aquesta capa no hi ha massa referències FW ja que és una capa independent de la infraestructura i que no requereix d'intèrprets per tal que l'usuari pugui visualitzar les dades.

### 2.1.3 Persistència

La capa de persistència és la que permet a l'aplicació mantenir les dades emmagatzemades. No necessàriament s'ha d'associar la capa de persistència amb una Bases de Dades, es pot generalitzar a altres sistemes com cues o inclús un servidor de correu electrònic. És tot allò que té a veure amb un accés a dades.

En aquest camp hi ha molts FW que han desenvolupat eines per a poder simplificar l'accés a les dades, aïllant d'aquesta manera el desenvolupador dels noms dels camps de les taules, o de les diferents restriccions d'integritat que es poden implementar en un sistema gestor de Bases de Dades, però inclús; intenten aïllar la capa de persistència del propi Gestor de Bases de Dades, és a dir: si tenim una aplicació X feta amb un FW de persistència, el fet de canviar de gestor de Bases de Dades, no ha de tenir més implicació que el fet de modificar els paràmetres de configuració del propi FW, sense necessitat de tocar més codi per tal que l'aplicació funcioni amb el nou Gestor de Bases de Dades.

### 2.1.3 Selecció de la capa d'estudi

La capa que selecciono per a realitzar l'estudi és la de persistència. Les raons principals que porten a aquesta decisió són:

- Facilitat d'avaluar en termes objectius la capa de persistència (memòria i velocitat)
- Extensió de documentació dels diferents FW de persistència actuals

## *2.2 Selecció dels FW a avaluar*

En aquest punt es mostren els diferents FW que s'han seleccionat per a realitzar l'estudi.

La selecció dels diferents FW s'ha fet tenint en compte les següents premisses:

**Premissa inicial:**

Han de ser gratuïts i de lliure distribució. No es podran utilitzar programes de pagament per a realitzar el projecte.

**JDO i JPA**

Primerament s'ha de tenir en compte que tant JPA com JDO són especificacions, mentre que els FW són les implementacions de les especificacions que s'utilitzen pels projectes.

Com a especificació cadascuna d'elles té les seves avantatges i inconvenients (per a veure el detall, anar a la pàgina del projecte Apache que les compara : http://db.Apache.org/jdo/jdo_v_jpa.html ).

### 2.2.1 Hibernate

Hibernate és un FW de persistència que està basat en el mapeig del model relacional amb objectes JAVA (mitjançant arxius XML, i actualment també mitjançant anotacions).

http://www.hibernate.org/

**Sobre Hibernate**

- Hibernate apareix l'any 2001, i s'uneix a JBOSS l'any 2003
- Conté documentació extensa
- Està funcionant sobre un conjunt molt important de projectes en producció, per exemple: jBpm, Andromeda, Confluence...
- Utilitza caché
- Té un llenguatge propi per a realitzar consultes a la Bases de Dades (HQL)
- No és un estàndard

La decisió d'utilitzar el FW Hibernate té a veure amb l'extensió i utilització d'aquest FW al llarg dels últims anys.

### 2.2.2 JDO

JDO API és una interfície estàndard basada en JAVA en el model de persistència d'abstracció. La versió JDO 2.0 (l'actual) està basada en l'especificació JAVA (JSR 243). A partir de la versió 2.0, el desenvolupament de l'API es fa dins del projecte Apache JDO.

http://java.sun.com/jdo/
http://db.Apache.org/jdo/javadoc.html

La implementació que s'utilitza pel "PersistenceManager" és: JPOX.

**Sobre JPOX:**

- JPOX apareix l'any 2003, i la primera release es publica l'any 2004
- Al Març del 2008 s'allibera la versió 1.2 de JPOX (la que s'utilitza per a aquest projecte)
- Llicència Apache 2
- És el FW de referència per a JDO
- Implementa l'especificació JDO 2.0 i JDO 2.1
- Requereix de re compilació (enhancer)
- Documentació (no molt extensa) sobre la utilització de les notacions de JDO

La decisió d'utilitzar el FW de JDO té a veure amb que JPOX és el FW de referència d'aquesta especificació.

### 2.2.2 Implementació JPA

No requereix de configuració XML (funciona amb anotacions).

JPA és una especificació que compta amb diferents implementacions. Per fer l'anàlisi es parteix de la implementació d'OpenJPA d'Apache (http://openjpa.Apache.org/).

Sobre OpenJPA:

- Llicència Apache 2
- Permet utilitzar anotacions
- Projectes que actualment utilitzen/suporten OpenJPA: ActiveMQ, BEA Weblogic Server, BEA Kodo, Spring...

La decisió d'utilitzar el FW de JPA té a veure amb que es vol provar una implementació basada en el projecte Apache.

http://java.sun.com/developer/technicalArticles/J2EE/jpa/

# 3. Definició de proves a realitzar

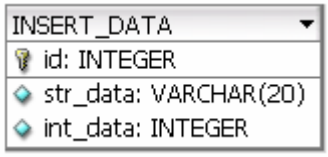
En aquest punt es defineixen les diferents proves que es realitzaran per a avaluar els diferents FW en les situacions que es descriuen en les proves.

## *3.1. Proves automatitzades*

### 3.1.1. Proves d'inserció de dades

Taules per a realitzar les proves:

**INSERT_DATA**: és una taula d'entrada buida, en el procés de proves s'omplirà amb dades.

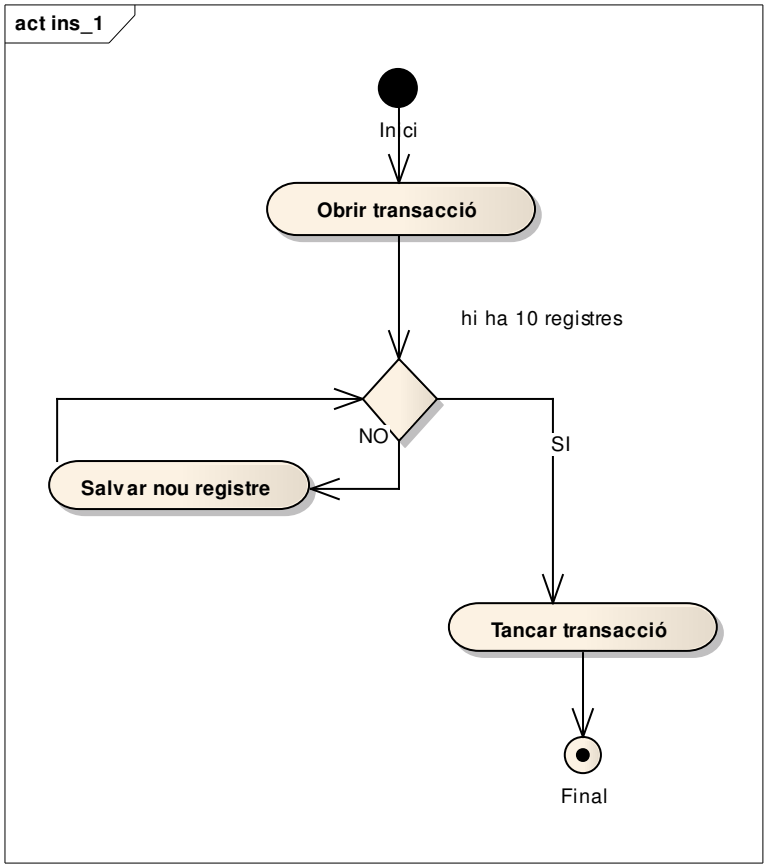


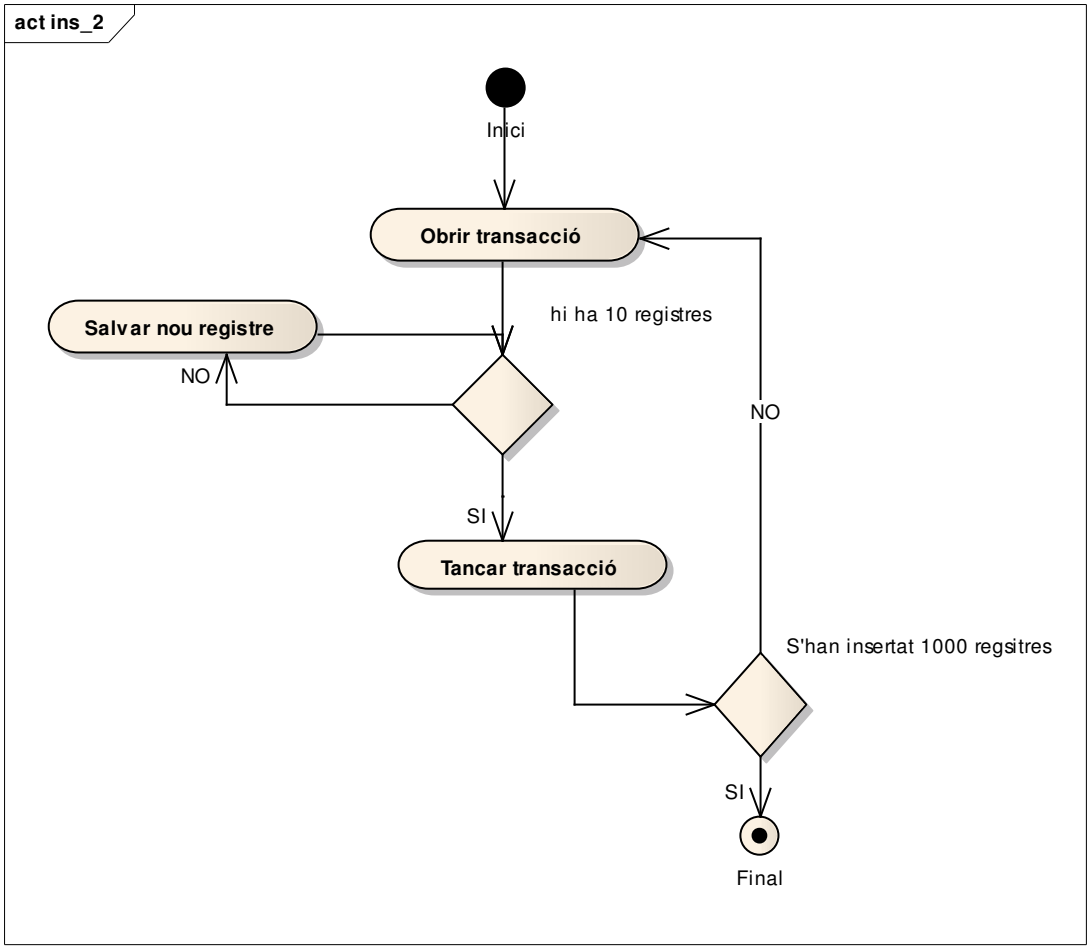

L'identificador és un INTEGER, que permet inserir fins a 4.294.967.295 registres.

Les proves d'inserció es basaran en dos tipus de proves:

- Proves d'inserció massiva de dades en una sola transacció
- Proves d'inserció en diferents transaccions

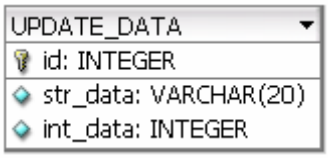
Proves a realitzar:

| **Identificador de la prova** | **ins_1** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat d'inserció en una sola transacció. |
| Metodologia | S'insereixen 100 registres en una sola transacció. |
| |  |

| **Identificador de la prova** | **ins_2** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat d'inserció en diferents transaccions. |
| Metodologia | S'insereixen 1.000 registres en diferents transaccions.<br>S'insereixen 10 registres per transacció. |
| |  |

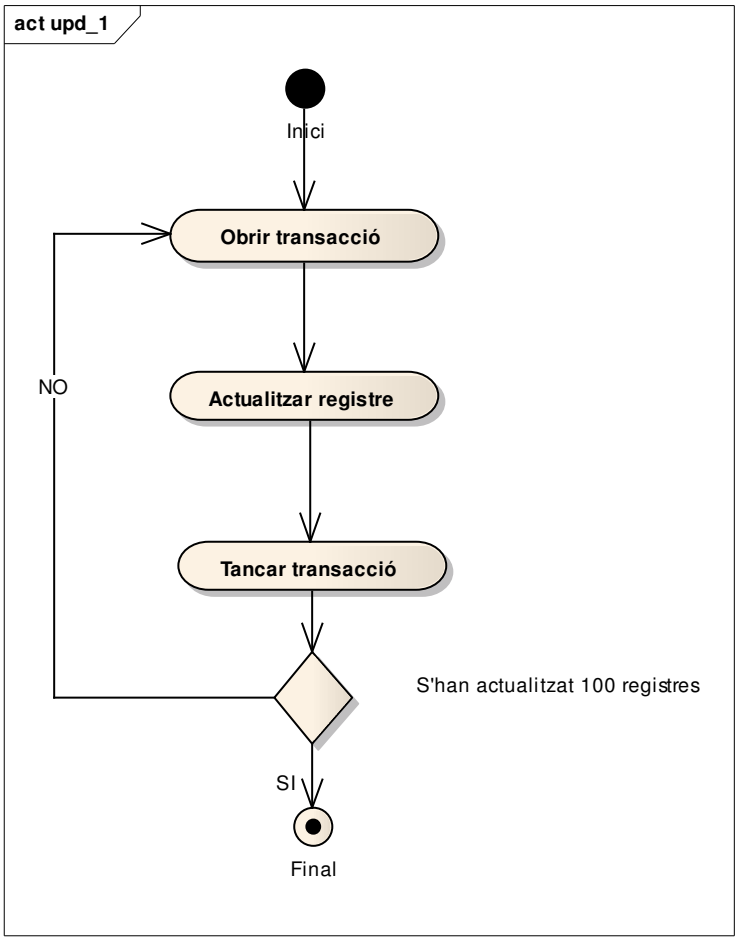
### 3.1.2. Proves d'actualització de dades

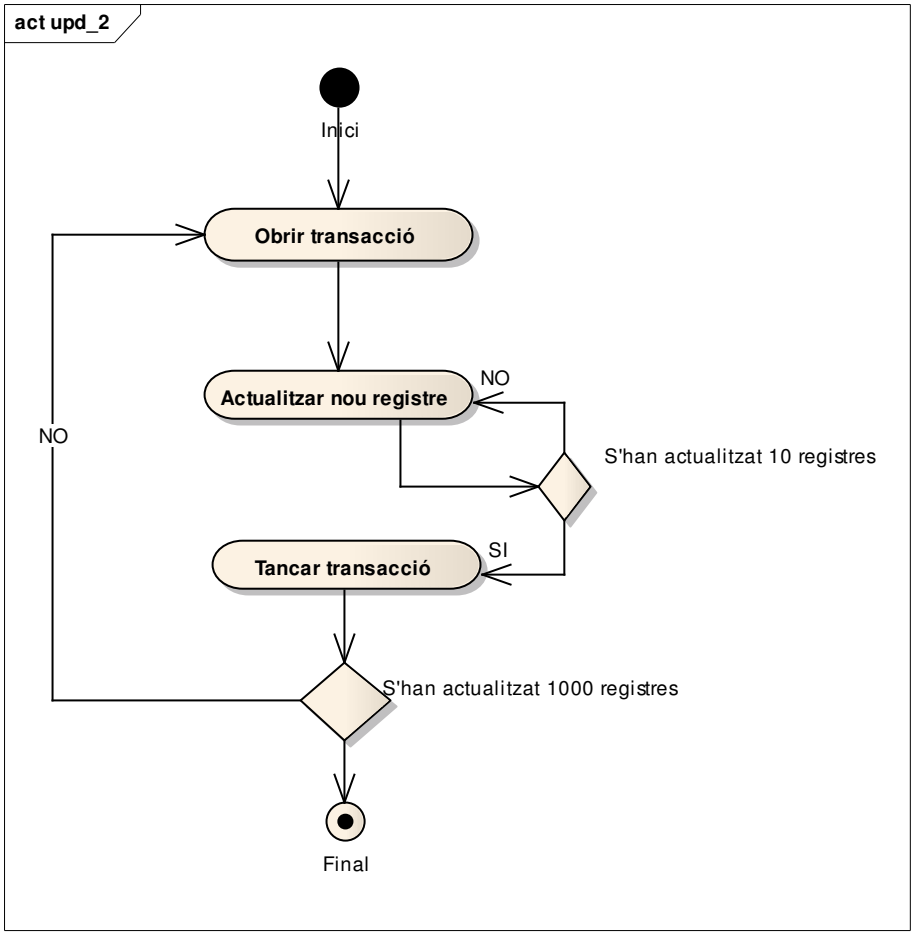
Taula amb les que es realitzen les proves d'actualització de dades:

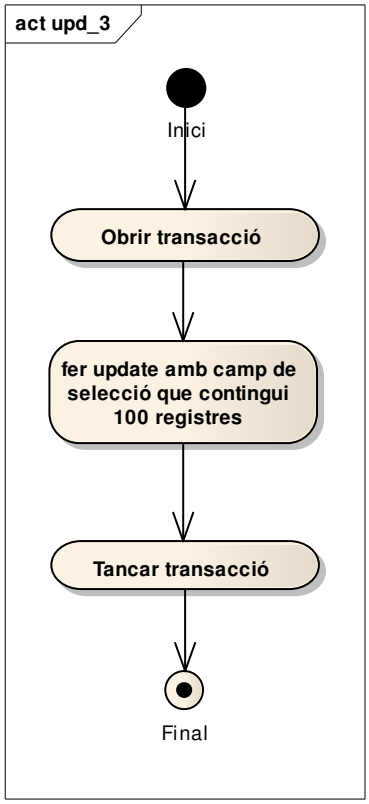



Conté el mateix tipus de dades que la taula d'inserció.

Proves a realitzar:

| **Identificador de la prova** | **upd_1** |
|---|---|
| Objectiu de la prova | Mesurar velocitat d'actualització de registres 1 a 1. |
| Metodologia | Actualització de 100 registres 1 a 1 utilitzant una sola transacció (100 transaccions d'una actualització de fila). |
| |  |

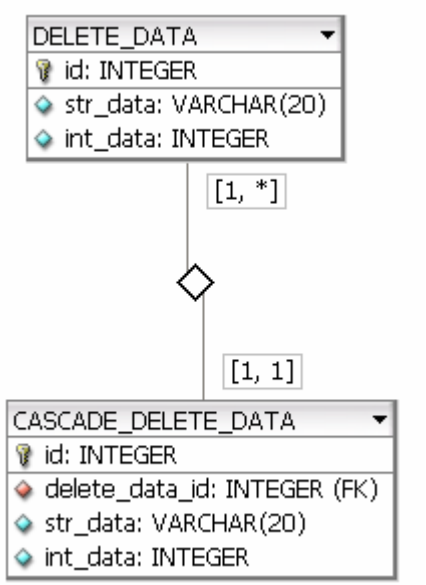
| **Identificador de la prova** | **upd_2** |
|---|---|
| Objectiu de la prova | Mesurar velocitat d'actualització de registres 10 a 10. |
| Metodologia | S'actualitzen 1.000 registres 10 a 10. |
| |  |

| **Identificador de la prova** | **upd_3** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat. |
| Metodologia | Actualització massiva en una sola transacció mitjançant un camp d'actualització que continguin 100 registres. |
| |  |

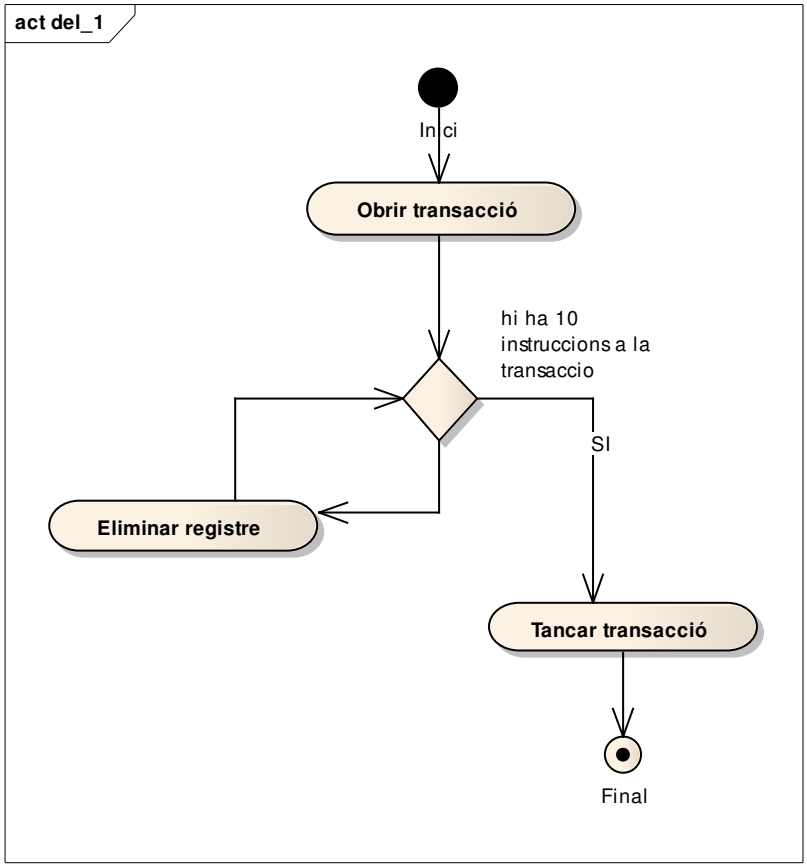
### 3.1.3. Proves d'eliminació de dades

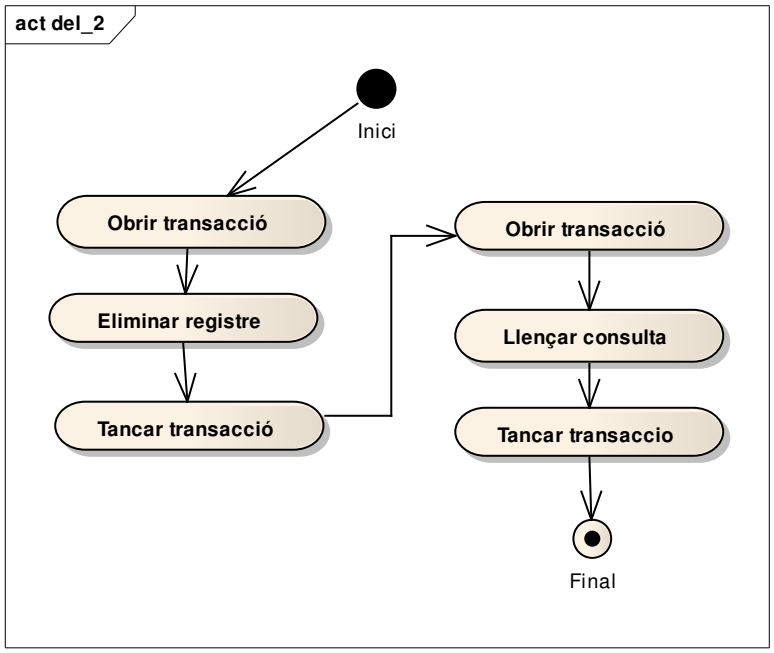
Les taules implicades en les proves són:

- **DELETE_DATA**: Conté dades d'eliminació
- **CASCADE_DELETE_DATA**: Conté dades que fan referència a la taula DELETE_DATA, i conté una constraint que indica que s'eliminen en cas de que en la taula pare s'hagin eliminat.



Proves a realitzar:

| **Identificador de la prova** | **del_1** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat. |
| Metodologia | Consulta que elimini els elements que estan a la taula **delete_data** però no a la taula **cascade_delete_data.** |
| |  |

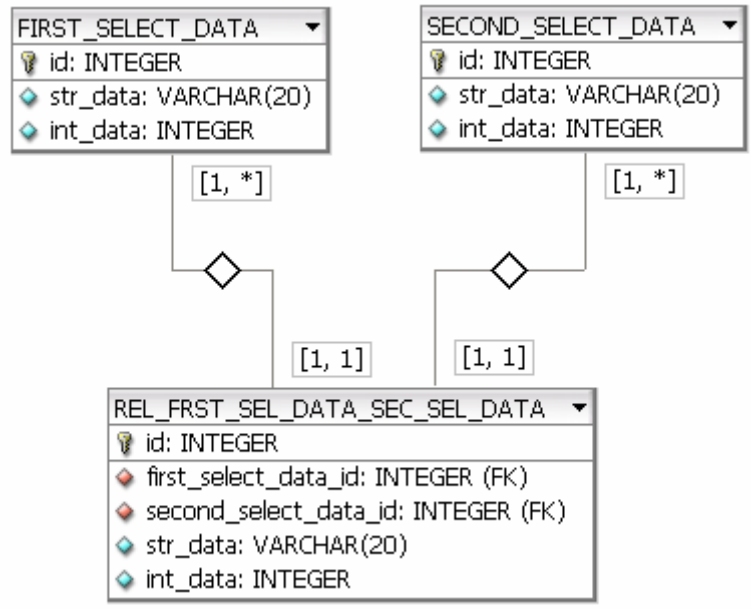
| **Identificador de la prova** | **del_2** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat i validar que s'eliminen les dades de **cascade_delete_data**. |
| Metodologia | Consulta que elimini elements que estan a la taula **delete_data** i també a la taula **cascade_delete_data.** |
| |  |
| Consulta d'exemple de validació | `SELECT `cascade_delete_data`.id, `cascade_delete_data`.delete_data_id FROM pfc.cascade_delete_data `cascade_delete_data` WHERE (`cascade_delete_data`.delete_data_id = 1)` |

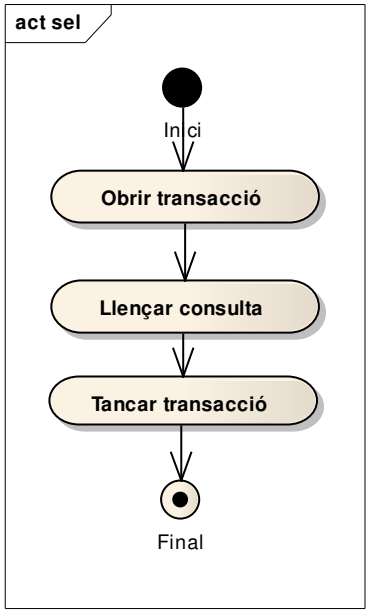
### 3.1.4. Proves de consulta de dades

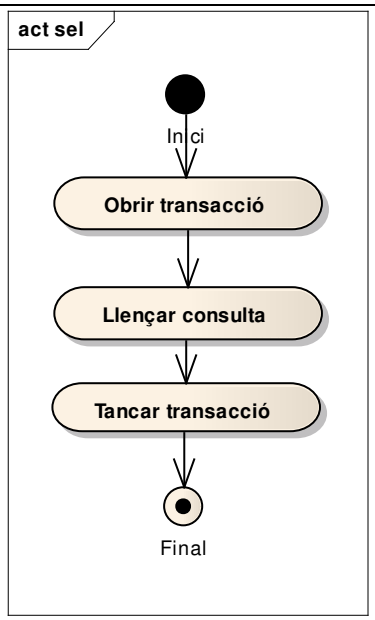
El que s'intenta avaluar a les proves de consulta es veure com carrega en memòria i la velocitat de les diferents consultes que es realitzen a la BBDD.

Per a realitzar aquestes proves s'utilitzaran les taules:

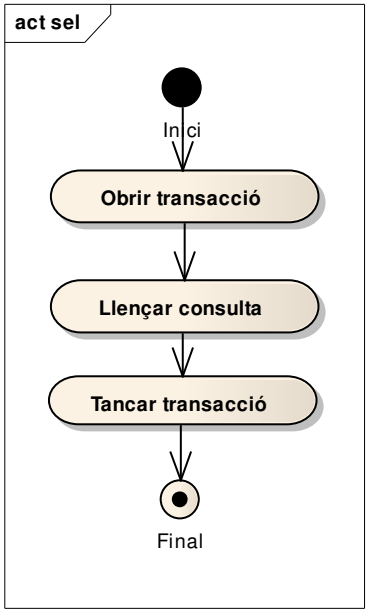
- **FIRST_SELECT_DATA**: 100.000 registres
- **SECOND_SELECT_DATA**: 100.000 registres
- **REL_FRST_SEL_DATA_SEC_SEL_DATA**: Conté 50.000 registres que són relacions entre la **FIRST_SELECT_DATA** i la **SECOND_SELECT_DATA** per tal de poder fer consultes utilitzant l'índex de FK



| **Identificador de la prova** | **sel_1** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat i el cost en memòria. |
| Metodologia | Consulta que contingui un resultat de 1.000 registres a la taula **first_selec_data.** |
| |  |
| Exemple de consulta | `SELECT` `first_select_data`.id,<br>`first_select_data`.str_data,<br>`first_select_data`.int_data<br>`FROM` pfc.first_select_data `first_select_data` |

| **Identificador de la prova** | **sel_2** |
|---|---|
| Objectiu de la prova | Mesurar la velocitat i el cost en memòria. |
| Metodologia | Consulta que contingui un resultat de 1.000 registres fent una JOIN entre un camp de la **first_select_data** i un de la **second_select_data**. |
| |  |

| | |
|---|---|
| Exemple de consulta | `SELECT `first_select_data`.id, `first_select_data`.str_data, `first_select_data`.int_data, `second_select_data`.str_data AS `str_data_1` FROM pfc.first_select_data `first_select_data`, pfc.second_select_data `second_select_data` WHERE (`first_select_data`.str_data = `second_select_data`.str_data)` |

| | |
|---|---|
| **Identificador de la prova** | **sel_3** |
| Objectiu de la prova | Mesurar la velocitat i el cost en memòria (veure si els objectes que s'instancien per la relació ocupen el mateix en memòria utilitzant els diferents FW). |
| Metodologia | Consulta sobre la **rel_frst_sel_data_sec_sel_data** per validar que els objectes relacionats no es carreguen en memòria en funció de les configuracions dels diferents FW. |
| |  |
| Exemple de consulta | `SELECT `rel_frst_sel_data_sec_sel_data`.id, `rel_frst_sel_data_sec_sel_data`. str_data, `rel_frst_sel_data_sec_sel_data`.first_select_data_id FROM ( pfc.rel_frst_sel_data_sec_sel_data `rel_frst_sel_data_sec_sel_data` INNER JOIN pfc.second_select_data `second_select_data` ON (`rel_frst_sel_data_sec_sel_data`.second_select_data_id = `second_select_data`.id)) INNER JOIN pfc.first_select_data `first_select_data` ON (`rel_frst_sel_data_sec_sel_data`.first_select_data_id = `first_select_data`.id)` |

## 3.2 Proves no automatitzades

| **Identificador de la prova** | **man_1** |
|---|---|
| Objectiu de la prova | Comparar la traducció a SQL de les diferents consultes. |
| Metodologia | Recuperació de dades indexades en la Bases de Dades i no indexades en la Bases de Dades mitjançant consultes amb diferents JOIN (veure com organitza les JOIN el FW per tal de fer la consulta). |

| **Identificador de la prova** | **man_2** |
|---|---|
| Objectiu de la prova | Facilitat de migració de gestor de Bases de Dades (configuració de l'entorn). |
| Metodologia | Donada una configuració amb un gestor de Bases de Dades concret, veure quins són els canvis necessaris per tal de poder canviar de gestor de Bases de Dades. |

| **Identificador de la prova** | **man_3** |
|---|---|
| Objectiu de la prova | Facilitat de migració de servidor d'aplicacions (configuració de l'entorn). |
| Metodologia | Donada una configuració amb un servidor d'aplicacions concret, veure quins són els canvis necessaris per tal de poder canviar de servidor d'aplicacions. |

| **Identificador de la prova** | **man_4** |
|---|---|
| Objectiu de la prova | Facilitat de codificació. |
| Metodologia | - Nombre d'arxius de configuració.<br>- Nombre d'objectes a manipular per a realitzar una acció de persistència.<br>- Facilitat de codificació i sincronització amb el model de dades (eines d'importació del model). |

| **Identificador de la prova** | **man_5** |
|---|---|
| Objectiu de la prova | Implementació de bloqueig optimista i pessimista. |
| Metodologia | - Validar que es suporten els dos tipus de bloqueigs.<br>- Validar si el bloqueig pessimista es realitza sobre la Bases de Dades o en memòria.<br>- Valorar la facilitat d'implementació basant-se en l'especificació del FW. |

## *3.3 Definir mesures de compliment d'especificació del propi Framework*

| **Identificador de la prova** | **espec_1** |
|---|---|
| Objectiu de la prova | Validar que l'especificació es compleix en els exemples proposats. |
| Metodologia | Comprovar que en la implementació de les diferents proves automatitzades es generen els objectes i les consultes que s'especifiquen a la documentació. |

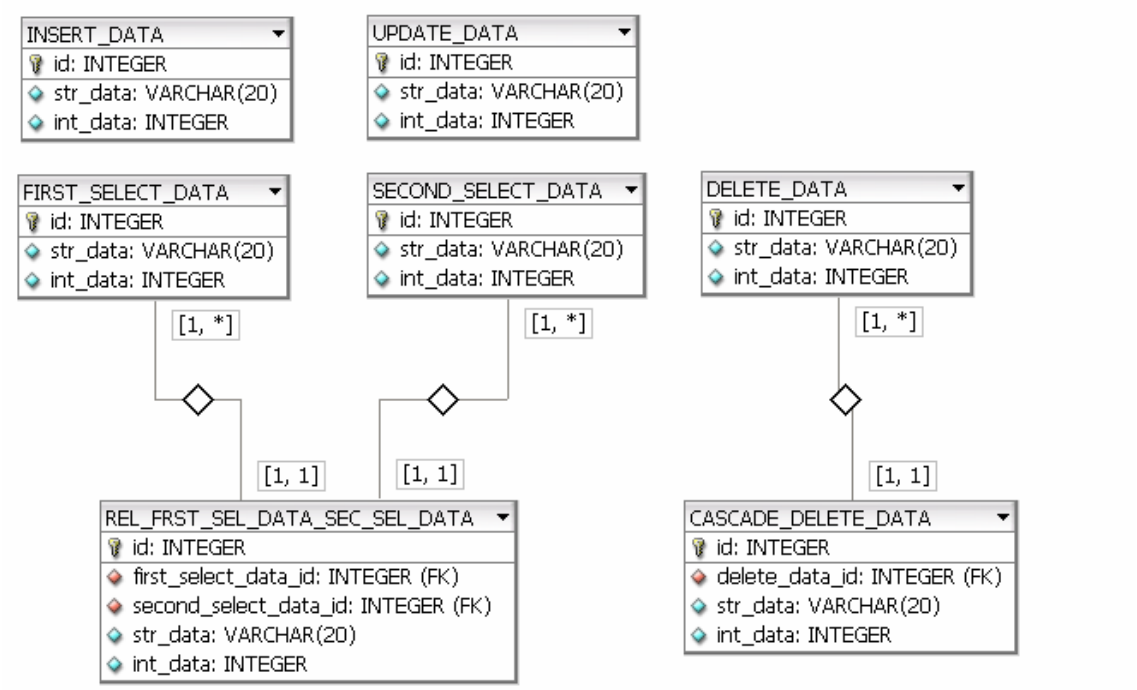
| **Identificador de la prova** | **espec_2** |
|---|---|
| Objectiu de la prova | Validar que l'especificació es compleix pel que fa a la independència del FW vs el sistema gestor de Bases de Dades. |
| Metodologia | Comprovar que si es canvia de gestor de Bases de Dades (el dialecte a utilitzar), les consultes generades compleixen amb les del nou gestor. |

| **Identificador de la prova** | **espec_3** |
|---|---|
| Objectiu de la prova | Validar que en els FW que es substitueix el XML per anotacions, és possible fer el mateix que permet fer el XML. |
| Metodologia | Comprovar que es poden utilitzar com a mínim els mateixos paràmetres i veure si n'apareixen de nous. |

## *3.4 Model de dades per a la realització de les proves*

### 3.4.1. Model

**INSERT_DATA**: taula d'insercions
**UPDATE_DATA**: taula d'actualitzacions
**FIRST_SELECT_DATA**: Taula de consulta 1
**SECOND_SELECT_DATA**: Taula de consulta 2
**REL_FRST_SEL_DATA_SEC_SEL_DATA**: Taula de relació de consultes 1 i 2
**DELETE_DATA:** Taula que conté dades a eliminar
**CASCADE_DELETE_DATA**: Taula que conté FK cap a la taula d'eliminació que s'eliminen automàticament a l'eliminar les de DELETE_DATA




Totes les taules contenen els camps:

str_data (camp de text de 20 caràcters)
int_data (camp de tipus enter de 4 bytes)

### 3.4.2. Script de creació de Bases de Dades

```
CREATE DATABASE IF NOT EXISTS pfc;
USE pfc;

-- ------------------------------------------------------------
-- Taula que conté les dades de inserció per a realització de proves.
-- ------------------------------------------------------------

CREATE TABLE INSERT_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id)
)
TYPE=InnoDB;

-- ------------------------------------------------------------
-- Conté dades de la segona taula de consulta
-- ------------------------------------------------------------

CREATE TABLE SECOND_SELECT_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id)
)
TYPE=InnoDB;

-- ------------------------------------------------------------
-- Conté les dades per a les proves d'actualització de dades.
-- ------------------------------------------------------------

CREATE TABLE UPDATE_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id)
)
TYPE=InnoDB;

-- ------------------------------------------------------------
-- Conté dades que serviran per a realitzar eliminacions.
-- ------------------------------------------------------------

CREATE TABLE DELETE_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id)
);

-- ------------------------------------------------------------
-- Conté dades de la primera taula de consulta
-- ------------------------------------------------------------

CREATE TABLE FIRST_SELECT_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id)
)
TYPE=InnoDB;

-- ---------------------------------------------------------
-- Conté dades per a realitzar eliminació en cascada
-- ---------------------------------------------------------

CREATE TABLE CASCADE_DELETE_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
delete_data_id INTEGER UNSIGNED NULL,
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id),
FOREIGN KEY(DELETE_DATA_id)
REFERENCES DELETE_DATA(id)
ON DELETE CASCADE
ON UPDATE NO ACTION
)TYPE=InnoDB;

-- ---------------------------------------------------------
-- Conté les dades de relació de les taules 1 i 2 de consulta.
-- ---------------------------------------------------------

CREATE TABLE REL_FRST_SEL_DATA_SEC_SEL_DATA (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
first_select_data_id INTEGER UNSIGNED NOT NULL,
second_select_data_id INTEGER UNSIGNED NOT NULL,
str_data VARCHAR(20) NULL,
int_data INTEGER UNSIGNED NULL,
PRIMARY KEY(id),
FOREIGN KEY(FIRST_SELECT_DATA_id)
REFERENCES FIRST_SELECT_DATA(id)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
FOREIGN KEY(SECOND_SELECT_DATA_id)
REFERENCES SECOND_SELECT_DATA(id)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)
TYPE=InnoDB;


-- ---------------------------------------------------------
-- Conté les dades per fer les proves de consultes entre SGBDS.
-- ---------------------------------------------------------

CREATE TABLE TEST_SGBD (
str_data VARCHAR(20) NULL,
int_data INTEGER NOT NULL,
PRIMARY KEY(int_data)
);
```

## *3.5 Entorn de desenvolupament*

Per a realitzar les proves automatitzades s'utilitza el següent entorn de desenvolupament.

- **Servidor d'aplicacions**: JBOSS 4.2.2 GA en cas que sigui necessari realitzar proves utilitzant un servidor d'aplicacions J2EE (aquesta necessitat s'analitzarà en funció de les proves realitzades de mesura automatitzada)
- **Versió de JDK**: 1.5_0_15
- **Servidor de Bases de Dades**: MySQL Server 5.0.51.
- **Programa:** Eclipse Europa.

# 4. Desenvolupament

## *4.1 Configuració de l'entorn per a realitzar les proves*

Tal i com s'indica a l'apartat 3.5, l'entorn de desenvolupament ha estat realitzat seguint els següents paquets i versions de programari.

### 4.1.1 Versió de JDK

**Versió de JDK:** 1.5_0_15.

La versió de JDK es descarrega de la web http://java.sun.com/javase/downloads/index_jdk5.jsp, i en el cas del projecte s'instal·la la versió per Windows.

### 4.1.2 Servidor de Bases de Dades

Tot i que en principi el servidor de Bases de Dades no ha de ser un condicionant per a fer les proves de persistència (els diferents FW que es volen valorar es presenten amb la capacitat d'interactuar amb diferents sistemes gestors de Bases de Dades) s'ha escollit MySQL com a servidor de Bases de Dades.

Una de les necessitats que es vol veure coberta és la capacitat de definir taules InnoDB que permeten definir regles en funció d'actualitzacions de FK.

Versió: MySQL Server 5.0.51 per Windows.

La pàgina web del projecte on es pot descarregar és.
http://dev.mysql.com/downloads/mysql/5.0.html#downloads

La instal·lació en Windows és assistida.

Dades de configuració de connexió:

Usuari: root
Password: root

Servidor de Bases de Dades addicional per a fer proves de portabilitat entre sistemes gestors de Bases de Dades: PostgreSQL Database Server 8.3. La instal·lació està basada en Windows i l'usuari i password de connexió són els mateixos.

### 4.1.2 Entorn de desenvolupament

**Eclipse**: L'entorn de desenvolupament que s'utilitza és Eclipse versió Europa (també descarregat des de la pàgina web: http://www.eclipse.org/).

Plugins que s'instal·len a Eclipse pel projecte:

**Plugins relacionats amb la capa de persistència:**

**JPOX Eclipse plugins:**

Es pot descarregar des de la pàgina del projecte de JPOX. Inclou funcionalitats per a recompilar bytecode.

**Hibernate Tools:**

Es pot descarregar directament utilitzant el sistema d'actualització de plugins propi d'Eclipse. Permet generar el model a partir de la Bases de Dades i té un editor de consultes HQL per a realitzar proves.

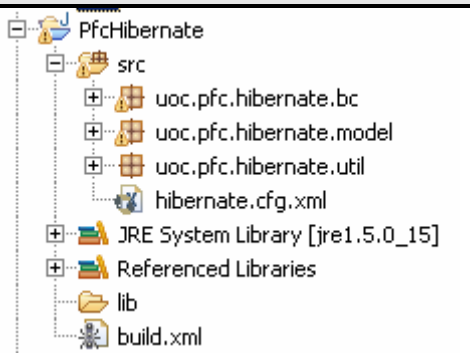
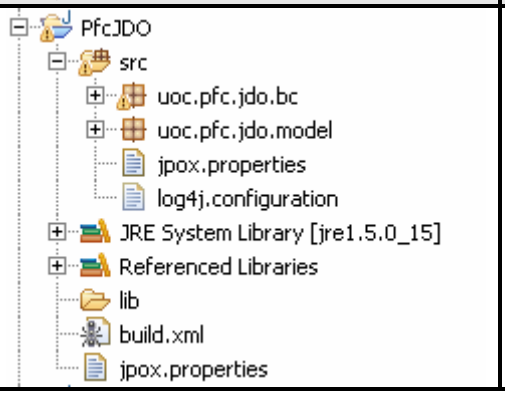
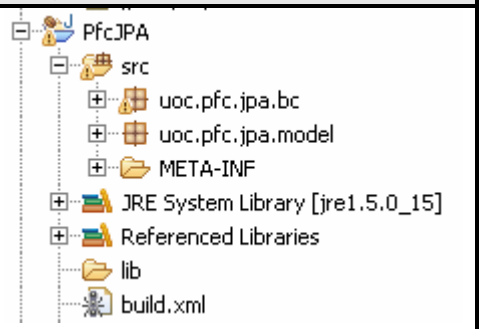
## *4.2 Definició d'estructura i de nomenclatures*

El desenvolupament està organitzat en tres projectes JAVA que mantenen la mateixa estructura de paquets:

| Nom del paquet | Descripció |
|---|---|
| Uoc.pfc.NNNNN.model | Es troben les classes generades a partir de la Bases de Dades. |
| Uoc.pfc.NNNNN.bc | Classes que executen el negoci de les diferents proves automatitzades (totes les classes que es troben dins d'aquest paquet acaben amb Bc)<br>- InsertBc<br>- UpdateBc<br>- SelectBc<br>- DeleteBc |
| Uoc.pfc.NNNNN.util | Classes d'utilitat. |

NNNNN pot ser segons el projecte: hibernate, jdo o jpa.

Esquema dels tres projectes:

| PfcHibernate | PfcJDO | PfcJPA |
|---|---|---|
|  |  |  |







Cadascun dels projectes conté un **ant** que incorpora dues tasques:

**ant model**: Aquesta tasca inicialitza els valors de les taules per a realitzar les proves (les taules han d'estar prèviament a la BBDD, no genera el model).

**ant exec**: Aquesta tasca executa les diferents proves que es descriuen en el conjunt de proves automàtiques.

## *4.3 Execució de les proves automàtiques*

### 4.3.1 Preparació de l'entorn d'execució

No és necessari configurar un entorn més enllà de la Bases de Dades i la capacitat d'executar programes codificats en JAVA.

[1] Instal·lar el servidor de Bases de Dades.

[2] Executar l'script SQL de creació de taules i carga de dades manual.

Script de creació amb les dades (aquest script carrega tant les dades com la definició de les taules):

```
mysql --user=root --password=root < RUTA\PFC_DMP_V1_1.sql
```

Script de creació sense dades (només creació de taules):

```
mysql --user=root --password=root < RUTA\PFC_DDL_V1_1.sql
```

[3] Execució de les tasques **ant** definides.

[3.1] ant buildAndExec: compila i executa les proves.
[3.2] ant exec: executa les proves.
[3.3] ant sgbd: executa les proves de la taula test_sgbd descrites a l'apartat de proves d'especificació per a diferents sistemes gestors de Bases de Dades.
[3.4] ant buildAndSgbd: executa les proves de la taula test_sgbd descrites a l'apartat de proves d'especificació per a diferents sistemes gestors de Bases de Dades.

Per a realitzar les proves de portabilitat entre sistemes gestors de Bases de Dades, executar la creació de la taula TEST_SGBD a l'entorn de PostgreSQL.

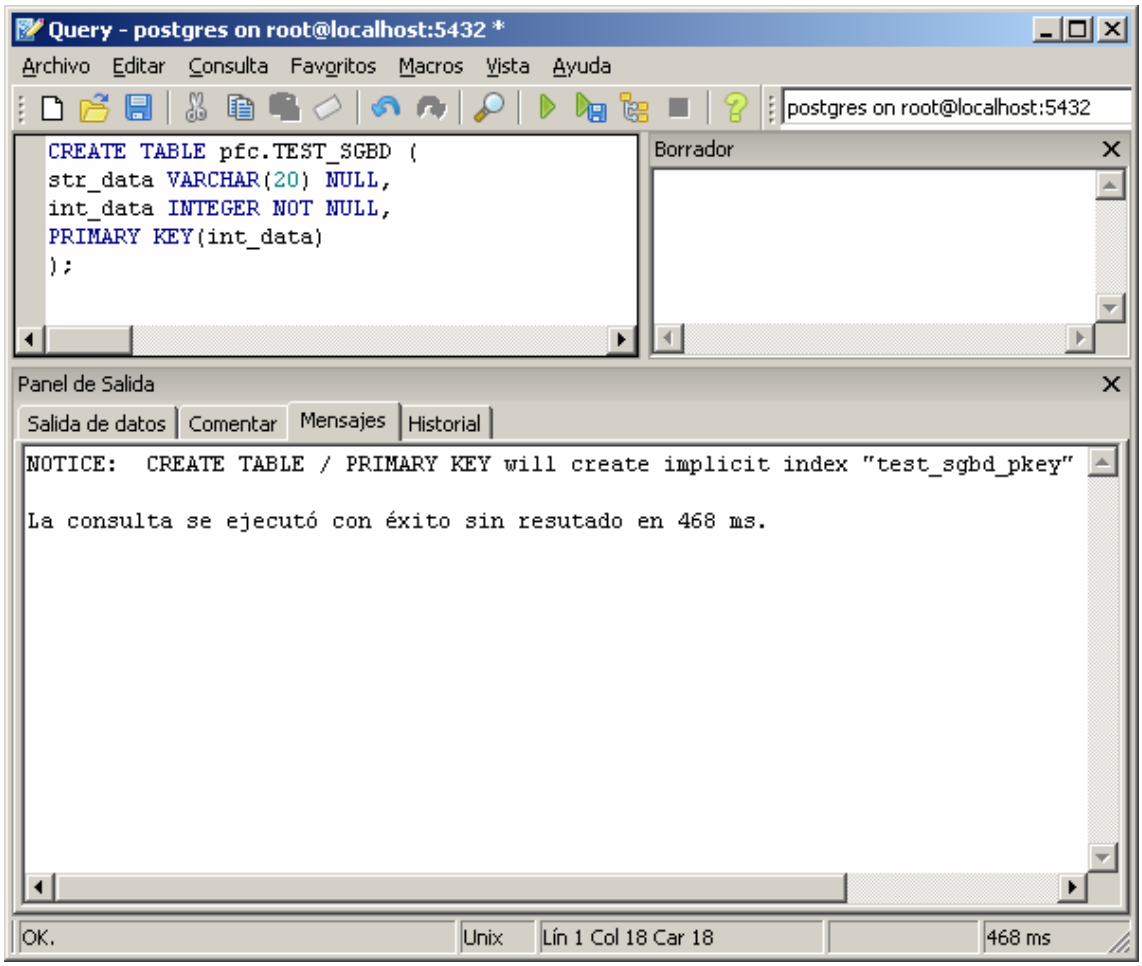
```
-- ------------------------------------------------------------
-- Conté les dades per fer les proves de consultes entre SGBDS.
-- ------------------------------------------------------------

CREATE TABLE TEST_SGBD (
str_data VARCHAR(20) NULL,
int_data INTEGER NOT NULL,
PRIMARY KEY(int_data)
);
```

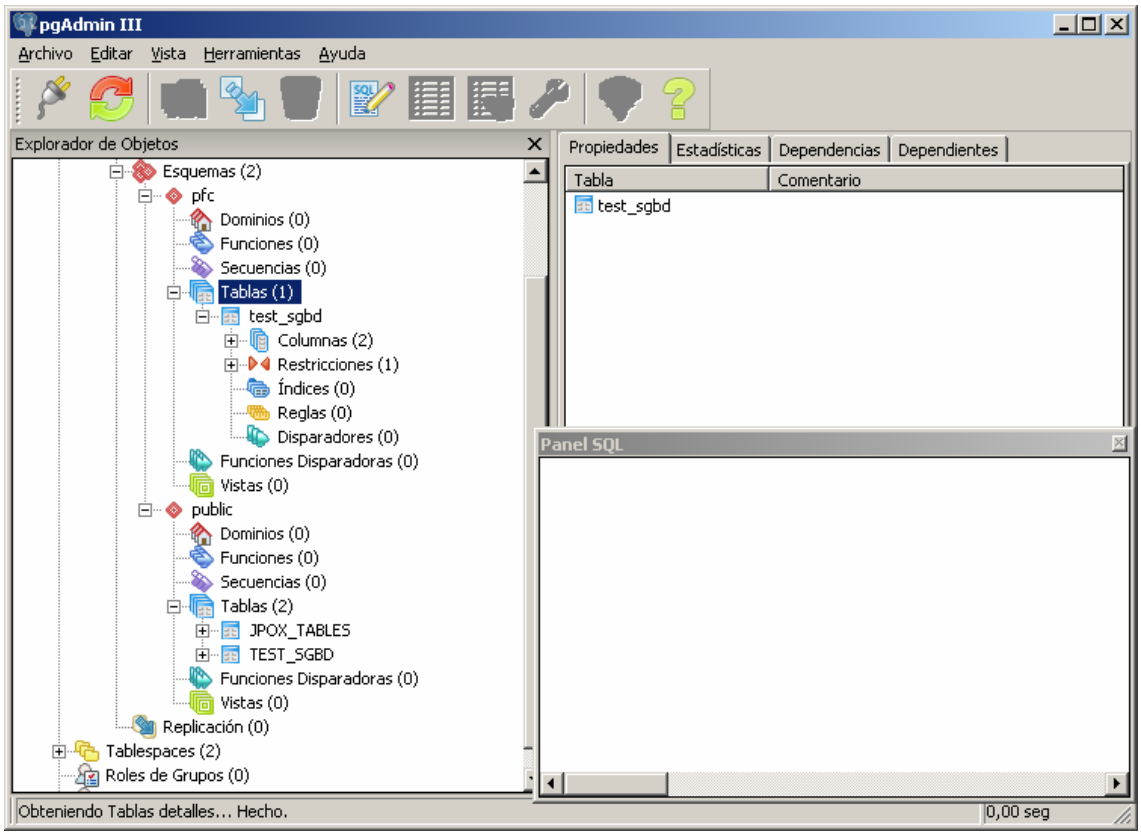
Un cop executat aquest script, la taula estarà llesta per ser consultada des dels diferents projectes executant les tasques d'ant definides als punts [3.3] i [3.4] segons si es vol o no compilar el projecte.

Cal recordar que per tal de que la configuració sigui acceptada per JPOX, aquesta taula s'ha de crear a la Bases de Dades PostgreSQL (esquema públic). En el cas dels altres FW es crea un nou esquema pfc i es crea la taula dins d'aquest.

Imatge de pgAdminIII:


Creació de la taula a PostgreSQL.






La taula al esquema públic la crea automàticament JPOX.

### 4.3.2 Preparació del model de dades

Executar la tasca **ant buildAndModel** (es recomana utilitzar el projecte que utilitza el FW Hibernate per a realitzar aquesta tasca ja que s'ha comprovat que és el més ràpid).

També es pot utilitzar el procediment partint dels SQL (scripts inclosos en la entrega)

Un cop executat aquest procés, la situació de la BBDD és la que es mostra a la taula:

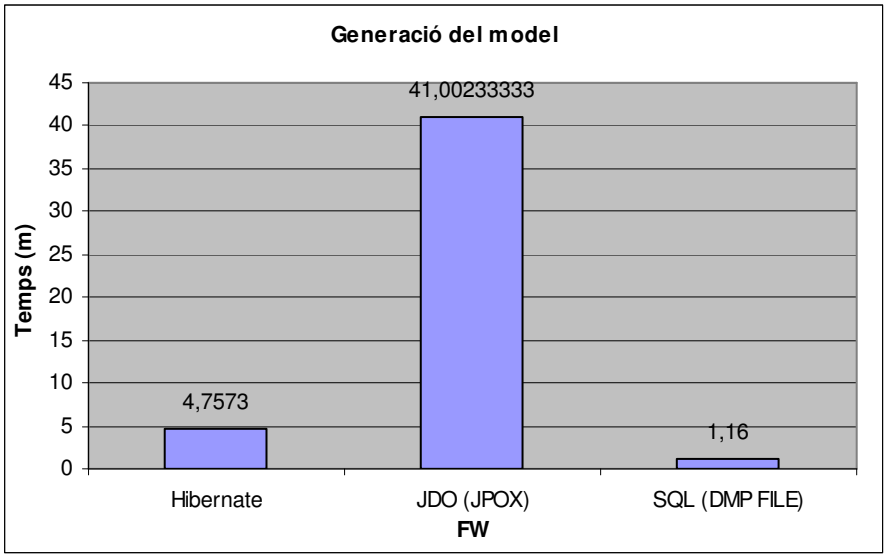
| Taula | Contingut |
|---|---|
| **INSERT_DATA** | No conté cap registre |
| **UPDATE_DATA** | Conté 1.100 registres<br><br>Registre del 1 al 100:<br>int_data = 0 fins a 99<br>Str_data= 1Tx_100_UPDT<br><br>Registre del 100 al 1.100:<br>int_data= 1.000 fins a 1.990 de 10 en 10 (el mateix en deu registres consecutius)<br>str_data= NTx_NNNN_UPDT (NNNN conté el mateix que int_data) |
| **FIRST_SELECT_DATA** | Conté 100.000 registres<br><br>Registre del 1 al 50.000:<br>int_data = 0 fins a 49.999<br>str_data = 1Tx_50000_SEL_1<br><br>Registre del 50.001 al 100.000:<br>int_data = 0 fins a 49.999<br>str_data = 1Tx_50000_RSEL_1 |
| **SECOND_SELECT_DATA** | Conté 100.000 registres<br><br>Registre del 1 al 50.000:<br>int_data = 0 fins a 49.999<br>str_data = 1Tx_50000_SEL_2<br><br>Registre del 50.001 al 100.000:<br>int_data = 0 fins a 49.999<br>str_data = 1Tx_50000_RSEL_2 |
| **REL_FRST_SEL_DATA_SEC_SEL_DATA** | Conté 50.000 registres: |

| | |
|---|---|
| | Registre del 1 al 50.000:<br>int_data = 0 al 49.999<br>str_data = 1Tx_50000_REL<br>first_select_data_id = 50.001 fins a 100.000<br>second_select_data_id = 50.001 fins a 100.000 |
| **DELETE_DATA** | Conté 100 registres:<br><br>Registre del 1 al 50:<br>int_data = 0 al 49<br>str_data = 1Tx_50_DEL<br><br>Registre del 51 al 100:<br>int_data = 50 al 99<br>str_data = 1Tx_50_CAS_DEL |
| **CASCADE_DELETE_DATA** | Conté 50 registres:<br><br>Registre del 1 al 50:<br>int_data = 0 al 49<br>str_data = 1Tx_50_CAS_DEL<br>delete_data_id = 51 fins a 100 |

#### 4.3.2.1 Temps d'execució dels diferents FW en la generació del model

Cal tenir en compte que l'execució de les proves s'ha realitzat mantenint el mateix nombre de registres per transacció en tots els FW analitzats.

El gràfic que es mostra a continuació mostra els temps d'execució de generar el model de dades amb els diferents FW.




#### 4.3.2.2 Comentaris respecte la generació del model

S'ha hagut de disminuir el nombre de registres per transacció en la inserció d'entrades a la taula FIRST_SELECT_DATA, SECOND_SELECT_DATA i REL_FRST_SEL_DATA_SEC_SEL_DATA.

Inicialment hi havia 1.000 registres per transacció, i actualment s'ha disminuït a 250 registres per transacció per tal que s'executin correctament les tasques de generació del model.

| FW | Capacitat mostrada en execució |
|---|---|
| **Hibernate** | 0 < x < 1.001 |
| **OpenJPA** | 0< x < 250 |
| **JPOX (JDO)** | 0< x < 1.001 |

Si es defineix un nombre petit de registres per transacció, el que acaba succeint és que amb OpenJPA es degrada l'execució i acaba fallant quan s'han inserit prop de 86.000 registres a FIRST_SELECT_DATA, SECOND_SELECT_DATA i 36.000 a REL_FRST_SEL_DATA_SEC_SEL_DATA.

Es modifica de nou l'execució del model per tal que carregui 1.000 registres per transacció. Es pot observar que el problema està reportat al projecte OpenJPA en les versions 1.0.1 i 1.0.2, i que actualment no està solucionat.

https://issues.Apache.org/jira/browse/OPENJPA-571

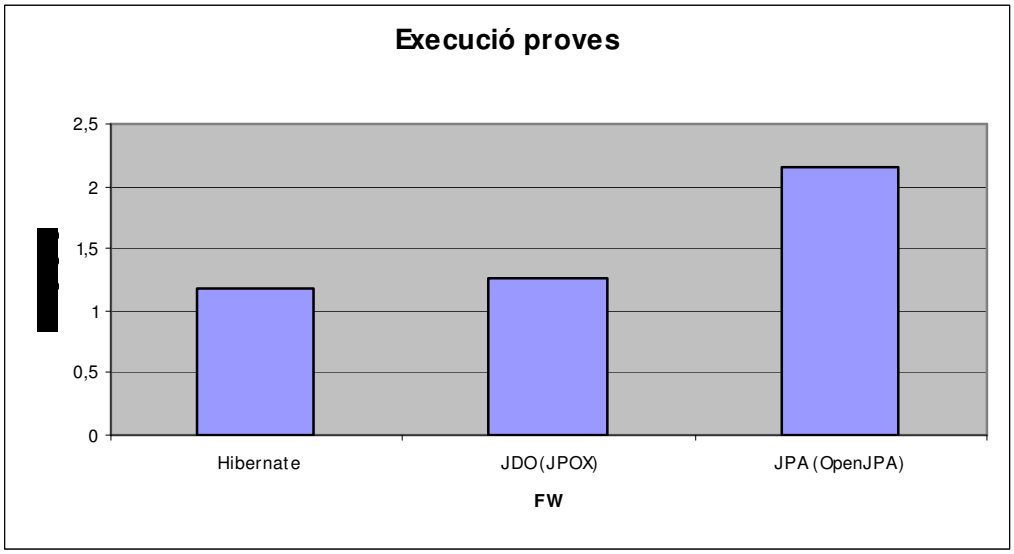
NOTA:

Aquest problema no afecta a l'execució de les proves automàtiques ja que de fet no hi ha transaccions que continguin un nombre tan elevat de registres a persistir.

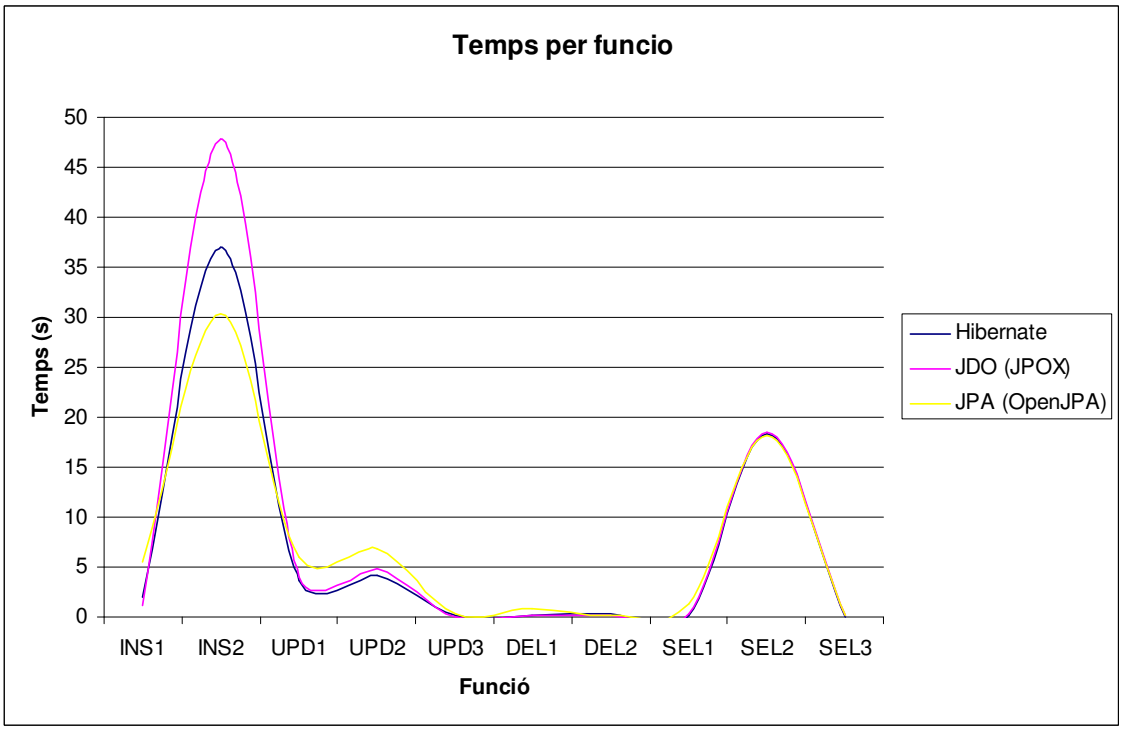
### 4.3.3 Execució de les proves automàtiques

L'execució de proves automàtiques consta de les proves definides a la PAC2.

Gràfica execució de temps:




Gràfica de temps per funció:




S'observa a la gràfica que a part de la inserció de molts registres les diferències de temps no són massa notables en l'execució dels diferents FW. En canvi, en processos massius d'inserció (generar model de dades) sí que s'observa una gran diferencia entre ells.

#### 4.3.3.1 Respecte l'execució de les proves automàtiques i el consum de memòria en el cas de la consulta SEL2 i la configuració per defecte

| FW | Carregat en memòria |
|---|---|
| **Hibernate** | NO (llença una select al consultar la relació) |
| **OpenJPA** | SI (no llença select al consultar la relació) |
| **JPOX (JDO)** | SI (no llença select al consultar la relació) |

Si Hibernate es configura en mode lazy la propietat relacionada (la clau forana), aleshores també ho carrega en memòria i no llença cap consulta.

### 4.3.4 Execució de les proves manuals

Les proves manuals es divideixen en dues metodologies d'execució:

[1] Aquelles que són una conseqüència directa de les proves automàtiques: En aquest cas es visualitzen els logs de les proves automàtiques per determinar el resultat de la prova manual.

[2] Les que tenen a veure amb les propietats de configuració: La prova consisteix en veure les diferències entre els diferents FW a partir de la pròpia documentació dels mateixos exemples recopilats.

| **Identificador de la prova** | **man_1** |
|---|---|
| Objectiu de la prova | Comparar la traducció a SQL de les diferents consultes. |

Amb la visualització dels log de les proves automatitzades s'observa que els SQL generats no contenen en cap dels FW diferències exceptuant el nom de les variables que utilitzen per a fer les traduccions.

**Exemple d'inserts:**

| **Hibernate** | **JDO (JPOX)** | **JPA(OpenJPA)** |
|---|---|---|
| insert into pfc.insert_data (int_data, str_data) values (?, ?) | SELECT `NEXT_VAL` FROM PFC.`SEQUENCE_TABLE` WHERE `SEQUENCE_NAME`=<'uoc.pfc.jdo.model.InsertData'> FOR UPDATE<br>INSERT INTO PFC.`INSERT_DATA` (`STR_DATA`,`INT_DATA`,`ID`) VALUES (<'NTx:_1150_INS'>,<1155>,<37806>)<br>UPDATE PFC.`SEQUENCE_TABLE` SET `NEXT_VAL`=(`NEXT_VAL`+<5>) WHERE `SEQUENCE_NAME`=<'uoc.pfc.jdo.model.InsertData'> | INSERT INTO pfc.insert_data (int_data, str_data) VALUES (?, ?) |
| | Cal tenir en compte que JPOX realitza una query addicional per tal de detectar quin és el següent número de seqüència amb la mateixa configuració de les PK, però s'observa a la insert que mentre a Hibernate i JPA només es passen dos valors per persistir, a JPOX es passa també el valor de l'ID.<br><br>També realitza un update de la taula de seqüències abans de tancar la transacció (en aquest cas ho fa +5). | |

NOTA:

És a dir, es detecta que tant Hibernate com JDO insereixen ordenadament les dades tal i com s'han realitzat (ordre) en el bucle, en canvi, JPA no ho fa, per tant els índex i int_data a vegades no concorden.

**Exemple de selects:**

| FW | SELECT | SELECT amb JOIN |
|---|---|---|
| Hibernate | select firstselec0_.id as id2_, firstselec0_.int_data as int2_2_, firstselec0_.str_data as str3_2_ from pfc.first_select_data firstselec0_ where firstselec0_.int_data>0 and firstselec0_.int_data<1001 | select firstselec0_.id as id2_, firstselec0_.int_data as int2_2_, firstselec0_.str_data as str3_2_ from pfc.first_select_data firstselec0_, pfc.second_select_data secondsele1_ where firstselec0_.int_data>0 and firstselec0_.int_data<1001 and secondsele1_.int_data=firstselec0_.int_data |
| | SELECT 'uoc.pfc.jdo.model.FirstSelectData' AS JPOXMETADATA,`THIS`.`ID`,`THIS`.`INT_DATA`,`THIS`.`STR_DATA` FROM PFC.`FIRST_SELECT_DATA` `THIS` WHERE `THIS`.`INT_DATA` > <0> AND `THIS`.`INT_DATA` < <1001> | select fsd.int_data from first_select_data fsd, second_select_data ssd where fsd.int_data>0 and fsd.int_data <1001 and fsd.int_data=ssd.int_data |
| | SELECT t0.id, t0.int_data, t0.str_data FROM pfc.first_select_data t0 WHERE (t0.int_data > ? AND t0.int_data < ?) | SELECT t0.id, t0.int_data, t0.str_data FROM pfc.first_select_data t0 CROSS JOIN pfc.second_select_data t1 WHERE (t0.int_data > ? AND t0.int_data < ? AND t1.int_data = t0.int_data) |

NOTA:

La diferència principal està en que el FW que utilitza JPA ha fet servir traduccions de JOIN, mentre que tant Hibernate com JPOX han utilitzat nomenclatura clàssica

.

**Exemple d'updates:**

| Hibernat | JDO (JPOX) | JPA |
|---|---|---|
| update pfc.update_data set int_data=?, str_data=? where id=? | UPDATE PFC.`UPDATE_DATA` SET `INT_DATA`=<0> WHERE `ID`=<234> | UPDATE pfc.update_data SET int_data = ?, str_data = ? WHERE id = ? |

**Exemple deletes:**

| Hibernate | JDO (JPOX) | JPA |
|---|---|---|
| delete from pfc.cascade_delete_data where id=? | DELETE FROM PFC.`DELETE_DATA` WHERE `ID`=<100> | DELETE FROM pfc.delete_data WHERE id = ? |

NOTA:

A l'observar els logs es demostra que tant Hibernate com JPOX (JDO) utilitzen la restricció d'eliminar (CASCADE_DELETE), mentre que JPA no ho realitza tal i com indica l'especificació.

http://db.Apache.org/jdo/jdo_v_jpa.html

| Identificador de la prova | man_2 |
|---|---|
| Objectiu de la prova | Facilitat de migració de gestor de Bases de Dades (configuració de l'entorn). |

Per tal de modificar l'accés a la Base de Dades caldrà canviar dues coses en qualsevol dels FW que s'estan utilitzant:

[1] El controlador (la llibreria que dóna accés a la BBDD).
[2] Fitxers de configuració o línies de codi on es defineixen les propietats de la connexió a la Base de Dades.

**Hibernate**

```
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost/pfc</property>
 <!--POSTGRESQL
<property name="hibernate.connection.url">jdbc:postgresql://localhost/pfc</property>
<property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
<property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property> -->
```

**JDO (JPOX)**

```
private void generateProperties(){
    Properties properties = new Properties();
    properties.setProperty("javax.jdo.PersistenceManagerFactoryClass","org.jpox.PersistenceManagerFa
    //properties.setProperty("javax.jdo.option.ConnectionDriverName","com.mysql.jdbc.Driver");
    //properties.setProperty("javax.jdo.option.ConnectionURL","jdbc:mysql://localhost/pfc");
    properties.setProperty("javax.jdo.option.ConnectionDriverName","org.postgresql.Driver");
    properties.setProperty("javax.jdo.option.ConnectionURL","jdbc:postgresql://localhost/postgres");
```

**JPA**

```
<property name="openjpa.ConnectionURL"
     value="jdbc:mysql://localhost:3306/pfc"/>
 <property name="openjpa.ConnectionDriverName"
     value="com.mysql.jdbc.Driver"/>
 <property name="openjpa.ConnectionUserName"
     value="root"/>
 <property name="openjpa.ConnectionPassword"
     value="root"/><!--
        <property name="openjpa.ConnectionURL"
     value="jdbc:postgresql://localhost/postgres"/>
 <property name="openjpa.ConnectionDriverName"
     value="org.postgresql.Driver"/> -->
```

**OpenJPA:**
Hi ha una carpeta config a l'arrel del projecte amb les dues configuracions del fitxer XML.

**JPOX:**
A la classe TestSgbdBc.java hi ha comentades les dues línies tal i com es mostra a la imatge.

**Hibernate:**
Hi ha una carpeta config a l'arrel del projecte amb les dues configuracions del fitxer XML.

| Identificador de la prova | man_3 |
|---|---|
| Objectiu de la prova | Facilitat de migració de servidor d'aplicacions (configuració de l'entorn). |

Per veure quins haurien de ser els canvis que s'han defectuar per tal de migrar d'un servidor de J2EE concret a un altre, agafo com a exemple els següents: JBOSS i BEA Weblogic.

En qualsevol cas el format de JNDI haurà de ser l'establert pel servidor J2EE.

Passos a realitzar per desplegar aplicació a servidor d'aplicacions J2EE:

[1] Delegar el control de transaccions al servidor d'aplicacions.
[2] Crear la connexió de Bases de Dades al servidor d'aplicacions (sigui JBOSS o BEA Weblogic).
[3] Localitzar el repositori a partir de JNDI.

**Hibernate**:

**Modificacions en codi: X**
**Modificacions descriptors: V**

**WEBLOGIC**

Modificar el fitxer hibernate.cfg.xml amb els valors que hi ha descrits a la documentació per a Weblogic amb JTA.

```
<!-- WebLogic transaction manager lookup (begin) -->
<property name="hibernate.transaction.manager_lookup_class">
org.hibernate.transaction.WeblogicTransactionManagerLookup</property>
<!-- WebLogic transaction manager lookup (end) -->
<!-- DataSource properties (begin) -->
<property name="hibernate.connection.datasource">NOM_JNDI</property>
<!-- DataSource properties (end) -->
 <property name="transaction.factory_class">
    org.hibernate.transaction.JTATransactionFactory
</property>
```

**JBOSS**

Modificacions al descriptor Hibernate.

Canviar el WeblogicTransactionManagerLookup per JBossTransactionManagerLookup respecte la configuració de Weblogic

**JPOX:**

**Modificacions en codi: V**
**Modificacions descriptors: V**

**WEBLOGIC**

[0] És necessari incorporar una nova llibreria jpox-jca-1.2.2.rar.
[1] Afegir un fitxer anomenat weblogic-ra.xml a la carpeta WEB-INF (el distribueix la pròpia llibreria):
    [1.1] En el fitxer ra.xml es defineixen les connexions a les BBDD.
    [1.2] S'ha de copiar el fitxer weblogic-ra.xml a META-INF (només en el cas de Weblogic).
[2] Desplegar a Weblogic.

Referència: http://www.jpox.org/docs/1_2/tutorials/j2ee.html#weblogic

**JBOSS**

[0] És necessari incorporar una nova llibreria jpox-jca-1.2.2.rar.
[1] Copiar la llibreria al directori "deploy" de JBOSS.
[2] Cal configurar una connexió a Bases de Dades mitjançant un descriptor al directori "deploy" de JBOSS, on s'informen les dades de connexió i el JNDI name (en l'exemple de la configuració el nom del JNDI és "jpox").

[3] En el codi, en el moment de recuperar la configuració de persistència es canvien les línies que recuperen el PersistenceManagerFactory per les següents:

```
InitialContext context=new InitialContext();
pmf=(PersistenceManagerFactory)context.lookup("java:/jpox");
```

Referència: http://www.jpox.org/docs/1_2/tutorials/j2ee.html#jboss4

**OpenJPA:**

**Modificacions en codi: X**
**Modificacions descriptors: V**

| **WEBLOGIC** |
| --- |
| En el cas d'OpenJPA no és necessari aplicar canvis, únicament és necessari treure les propietats locals i afegir la línia que determina el JTA datasource.<br><br>La línia té aquest format: <jta-data-source>JNDI</jta-data-source>.<br><br>No és necessari especificar el provider perquè el proveïdor per defecte de Weblogic és OpenJPA + Kodo.<br><br>Referències:<br>http://openjpa.Apache.org/integration.html<br>http://forums.bea.com/forum.jspa?forumID=500000029<br>http://dev2dev.bea.com/blog/jbayer/archive/2008/01/workshop_jpa_an.html |
| **JBOSS** |
| Cal modificar dues línies del fitxer de configuració de persistence.xml:<br><br><provider>org.Apache.openjpa.persistence.PersistenceProviderImpl</provider><br><jta-data-source>java:/jdbc/JNDINAME</jta-data-source> |

| **Identificador de la prova** | **man_4** |
|---|---|
| Objectiu de la prova | Facilitat de codificació. |
| Metodologia | - Nombre d'arxius de configuració<br>- Nombre d'objectes a manipular per a realitzar una acció de persistència<br>- Facilitat de codificació i sincronització amb el model de dades (eines d'importació del model) |

Per valorar el nombre d'arxius i el nombre d'objectes a manipular per a realitzar una acció de persistència es mostra la següent taula:

| | **Hibernate** | **JPOX** | **OpenJPA** |
|---|---|---|---|
| **Nombre arxius configuració** | 1 | 1 | 1 |
| **Nombre d'objectes a manipular per a realitzar una acció de persistència** | Session<br>Transaction<br>Objectes del model | PersistenceManager<br>TransactionManager<br>Objectes del model | EntityManager<br>EntityTransaction<br>Objectes del model |
| **Es disposa d'eines per a importar el model de Bases de Dades?** | SI (HibernateTools) | SI (plugin d'Eclipse JPOX o tasca ant). | SI (no directament OpenJPA però hi ha múltiples eines que ho fan) |

En quant a la facilitat de codificació entre els diferents FW, hi ha una diferència notable entre Hibernate i OpenJPA envers a JPOX. JPOX requereix de la compilació de bytecode per a la seva execució, això implica diferents tasques que caldrà tenir en compte a la hora de treballar amb aquest FW:

[1] Si es treballa sense cap mena d'entorn (per exemple amb Notepad i compilant amb ant) caldrà afegir una tasca addicional abans d'executar el programa que compili aquest byte.
[2] Si es treballa amb un entorn (per exemple Eclipse) aleshores JPOX posa a la disposició plugins que permeten solucionar el problema integrant-se amb l'auto-build de l'entorn.

| **Identificador de la prova** | **man_5** |
|---|---|
| Objectiu de la prova | Implementació de bloqueig optimista i pessimista. |
| Metodologia | - Validar que es suporten els dos tipus de bloqueigs.<br>- Validar si el bloqueig pessimista es realitza sobre la Bases de Dades o en memòria.<br>- Valorar facilitat de la implementació basat en l'especificació del FW. |

A la taula següent es mostren els tipus de bloqueigs suportats per els FW i com s'implementen:

| | **Hibernate** | **JPOX** | **OpenJPA** |
|---|---|---|---|
| **Suporten els dos tipus de bloqueig?** | SI | SI | SI |
| **El bloqueig pessimista es realitza sobre BBDD o en memòria.** | Bases de Dades (veure punt 11.4 del chapter 11 on a la primera línia diu: "*Hibernate will always use the locking mechanism of the database, never lock objects in memory!*")<br>Ofereix diferents tipus de bloqueig en el moment de llençar la consulta. | Bases de Dades, si s'especifica el paràmetre, aleshores canvia les selects per afegir un SELECT FOR UPDATE. | Bases de Dades, modifica les selects (funcionament com JPOX) |

Canvis d'implementació basat en l'especificació del FW:

**JPOX:**
Per a determinar el tipus de bloqueig que es vol utilitzar, només cal especificar-ho en "org.jpox.rdbms.useUpdateLock" en valor "true".

**OpenJPA:**
Es troba a la propietat: openjpa.Optimistic.

**Hibernate:**
Es pot definir en el moment de realitzar una consulta a un objecte. Té algun mode de bloqueig específic per a Oracle (per exemple: LockMode.UPGRADE_NOWAIT).

Referències:
http://openjpa.Apache.org/docs/latest/manual/ref_guide_conf_openjpa.html#openjpa.Optimistic
http://www.jpox.org/docs/1_2/javadocs/core/org/jpox/util/ReadWriteLock.html
http://www.jpox.org/docs/1_2/transactions.html
http://www.jpox.org/docs/1_2/jdo_orm/versioning.html
http://www.hibernate.org/hib_docs/reference/en/html/transactions.html

### 4.3.5 Mesures d'especificació

| **Identificador de la prova** | **espec_1** |
|---|---|
| Objectiu de la prova | Validar que l'especificació es compleix en els exemples proposats. |
| Metodologia | Comprovar que la implementació de les diferents proves automatitzades generen els objectes i les consultes que s'especifiquen a la documentació. |

Tal i com estan configurats els diferents FW es valida que:

- La creació d'objectes en memòria correspon al que s'especifica a la documentació. Una demostració de que no es carreguen més objectes dels que es demanen és el realitzat en la prova automàtica on es comprova que segons la configuració s'obliga al FW a realitzar una altre select per a recuperar un objecte relacionat o no fer-ho perquè ja el té en memòria.

- Es generen les consultes que es mostren a la documentació. En la prova manual s'observa quines són les diferents consultes que es llencen per a select, update, delete i insert, i estan d'acord amb la documentació. L'únic FW que realitza accions addicionals que no s'especifiquen per codi és JPOX (que prepara una taula per mantenir les seqüències i li fa un update un cop ha inserit el registre). Tot i que això apareix a la documentació no s'especifica per codi de forma explícita.

| **Identificador de la prova** | **espec_2** |
|---|---|
| Objectiu de la prova | Validar que l'especificació es compleix pel que fa a la independència del FW vs el sistema gestor de Bases de Dades. |
| Metodologia | Comprovar que si es canvia de gestor de Bases de Dades (el dialecte a utilitzar), les consultes generades compleixen amb les del nou gestor. |

Cal tenir en compte, que si s'han utilitzat propietats específiques d'un gestor de Bases de Dades, com per exemple, en el cas actual l'autonumèric de MySQL, en el moment de migrar s'hauran de solucionar aquestes situacions, tot i que això és degut a que no s'han utilitzat propietats que admetin tots els sistemes gestors de Bases de Dades, independentment del FW a utilitzar. JPOX en canvi té una taula de seqüències pròpia que ens permet evitar aquest problema.

Per fer la prova en el nostre cas es defineix una nova taula anomenada TEST_SGBD sobre la que es farà un insert, un update, un select i un delete.

Script de creació de la taula per fer les proves.

```
CREATE TABLE TEST_SGBD (
str_data VARCHAR(20) NULL,
int_data INTEGER NOT NULL,
PRIMARY KEY(int_data)
);
```

Es defineix una classe del model d'acord amb la taula i un nou Bc que fa un insert, un update, un select i un delete.

Es creen dues tasques noves als arxius ant:

- sgbd: executa les proves sense compilar
- buildAndSgbd: compila i executa les proves

Cal tenir en compte que per canviar de servidor de Bases de Dades s'han de seguir els passos descrits en el punt man_2 on s'explica quins són els arxius que s'han de modificar per canviar de servidor.

**JPOX (Nota important: per aconseguir que funcioni amb PostgreSQL, s'ha de realitzar sobre el esquema públic, referència:** http://www.jpox.org/servlet/jira/secure/ReleaseNote.jspa?projectId=10000&styleName=Html&version=10201**)**

La diferència entre un i altre és que canvia el caràcter ` per “

| | MySQL | PostgreSQL |
|---|---|---|
| INSERT | INSERT INTO PFC.`TEST_SGBD` (`STR_DATA`,`INT_DATA`) VALUES (<'1'>,<1>) | INSERT INTO "TEST_SGBD" ("STR_DATA","INT_DATA") VALUES (<'1'>,<1>) |
| UPDATE | UPDATE PFC.`TEST_SGBD` SET `STR_DATA`=<'2'> WHERE `INT_DATA`=<1> | UPDATE "TEST_SGBD" SET "STR_DATA"=<'2'> WHERE "INT_DATA"=<1> |
| SELECT | SELECT 'uoc.pfc.jdo.model.TestSgbd' AS JPOXMETADATA,`THIS`.`INT_DATA`,`THIS`.`STR_DATA` FROM PFC.`TEST_SGBD` `THIS` WHERE `THIS`.`INT_DATA` = <1> | SELECT 'uoc.pfc.jdo.model.TestSgbd' AS JPOXMETADATA,"THIS"."INT_DATA","THIS"."STR_DATA" FROM "TEST_SGBD" "THIS" WHERE "THIS"."INT_DATA" = <1> |
| DELETE | DELETE FROM PFC.`TEST_SGBD` WHERE `INT_DATA`=<1> | DELETE FROM "TEST_SGBD" WHERE "INT_DATA"=<1> |

**OpenJPA:** No s'observen diferències (en la visualització prèvia)

| | MySQL | PostgreSQL |
|---|---|---|
| INSERT | INSERT INTO pfc.test_sgbd (int_data, str_data) VALUES (?, ?) | INSERT INTO pfc.test_sgbd (int_data, str_data) VALUES (?, ?) |
| UPDATE | UPDATE pfc.test_sgbd SET str_data = ? WHERE int_data = ? | UPDATE pfc.test_sgbd SET str_data = ? WHERE int_data = ? |
| SELECT | SELECT t0.int_data, t0.str_data FROM pfc.test_sgbd t0 WHERE (t0.int_data =?) | SELECT t0.int_data, t0.str_data FROM pfc.test_sgbd t0 WHERE (t0.int_data = ?) |
| DELETE | DELETE FROM pfc.test_sgbd WHERE int_data = ? | DELETE FROM pfc.test_sgbd WHERE int_data = ? |

**Hibernate:** No s'observen diferències(en la visualització prèvia)

| | Hibernate | JPOX |
|---|---|---|
| INSERT | insert into pfc.TEST_SGBD (str_data, int_data) values (?, ?) | insert into pfc.TEST_SGBD (str_data, int_data) values (?, ?) |
| UPDATE | update pfc.TEST_SGBD set str_data=? where int_data=? | update pfc.TEST_SGBD set str_data=? where int_data=? |
| SELECT | select testsgbd0_.int_data as int1_0_0_, testsgbd0_.str_data as str2_0_0_ from pfc.TEST_SGBD testsgbd0_ where testsgbd0_.int_data=? | select testsgbd0_.int_data as int1_0_0_, testsgbd0_.str_data as str2_0_0_ from pfc.TEST_SGBD testsgbd0_ where testsgbd0_.int_data=? |
| DELETE | delete from pfc.TEST_SGBD where int_data=? | delete from pfc.TEST_SGBD where int_data=? |

| **Identificador de la prova** | **espec_3** |
|---|---|
| Objectiu de la prova | Validar que en els FW que es substitueix el XML per anotacions, és possible fer el mateix que permet fer el XML. |
| Metodologia | Comprovar que es poden utilitzar com a mínim els mateixos paràmetres i veure si n'apareixen de nous. |

Limitacions conegudes a anotacions: És necessari JDK 1.5

**JPOX:**

En la documentació de JPOX es recomana no utilitzar únicament anotacions o meta data, sinó que aquelles propietats que tenen a veure amb els camps de les taules o que han de ser reconegudes fora del temps de desplegament, estiguin en anotacions; mentre que aquelles propietats que es mantenen invariables es configurin en els arxius de meta data.

Existeix un llistat de propietats d'anotacions i un altre de meta data per la implementació de JDO 2.1.

Llistat d'anotacions: http://www.jpox.org/docs/1_2/jdo/annotations.html
Llistat de meta data: http://www.jpox.org/docs/1_2/jdo/metadata.html

Es pot arribar a utilitzar el sistema sense configurar cap tipus de meta data, més enllà de les definicions de les classes de persistència.

**OpenJPA:**

En la documentació d'OpenJPA no s'especifica quines són les anotacions suportades, se sobreentén que són aquelles que ofereix la pròpia especificació de Sun sobre JPA.

Referència: http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html

En el document de les especificacions es mostren les diferents dades que poden ser definides en un XML i les que poden ser definides a nivell d'anotació.

Segons l'especificació, els atributs que es defineixen a nivell de XML (meta data) serveixen per sobreescriure aquells que estan definits en les anotacions.

No és necessari mantenir XML per a JPA: es pot treballar únicament amb anotacions.

**Hibernate:**

Hibernate inicialment funcionava a partir d'arxius XML, per tal de fer les transformacions, però actualment permet utilitzar les anotacions descrites a JDK 1.5 en substitució dels XML. També aporta anotacions pròpies que permeten realitzar funcions especials.
En la documentació es diu que es pot funcionar sense especificar cap fitxer XML per a les transformacions entre les classes i el model, i durant aquest projecte s'ha realitzat d'aquesta manera.

Referències:
http://www.hibernate.org/397.html
http://java.sun.com/products/ejb/docs.html

# 5. Conclusions

## *5.1 Proves realitzades i resultats*

Abans d'extraure qualsevol conclusió, cal dir que els tres FW que he utilitzat per a fer les proves tenen avantatges i inconvenients, i que algunes d'aquestes característiques, no venen determinades pel FW en concret, sinó per l'especificació que implementa (per exemple: el fet que OpenJPA no realitzi funcions en cascada, no es pot atribuir a que el FW no ho faci, sinó a que la pròpia especificació de JPA no ho preveu).

Així doncs, la comparació es fa estrictament sobre els FW, independentment de les especificacions que implementin.

Per a omplir la taula de comparació utilitzo els següents valors:

| Valor | Significat |
|---|---|
| 2 | Simplifica o fa que l'execució sigui més àgil o fiable. |
| 1 | Es mou dins d'uns marges que fan que la implementació amb el FW sigui viable. |
| 0 | Es pot millorar significativament. |
| -1 | Situació complicada de difícil resolució que dificulta la utilització en el cas concret. |

Les proves automàtiques han permès fer una valoració objectiva de temps, sempre executant sobre el mateix PC i amb el model de dades restaurat a cada execució.

| Proves automàtiques | Hibernate | JPOX | OpenJPA |
|---|---|---|---|
| Temps execució del model | 2 | 1 | -1[1] |
| INS1 | 1 | 1 | 1 |
| INS2 | 1 | 1 | 1 |
| UPD1 | 1 | 1 | 1 |
| UPD2 | 1 | 1 | 1 |
| UPD3 | 1 | 1 | 1 |
| DEL1 | 1 | 1 | 1 |
| DEL2 | 1 | 1 | 1 |
| SEL1 | 1 | 1 | 1 |
| SEL2 | 1 | 1 | 1 |
| SEL3 | 1 | 1 | 1 |
| **Resultat** | **12** | **11** | **9** |

Les proves manuals permeten veure amb més detall que és allò que està passant quan s'executa una prova automàtica o bé determinar quins seran els canvis que s'hauran de realitzar en cas que es vulgui per exemple canviar de Sistema Gestor de Bases de Dades.

| Proves manuals | Hibernate | JPOX | OpenJPA |
|---|---|---|---|
| man_1 | 1 | 1 | 0[2] |
| man_2 | 1[3] | 1[4] | 1[5] |
| man_3 | 1 | 1 | 1 |
| man_4 | 1 | 0[6] | 1 |
| man_5 | 2[7] | 1 | 1 |
| **Resultat** | **6** | **4** | **4** |

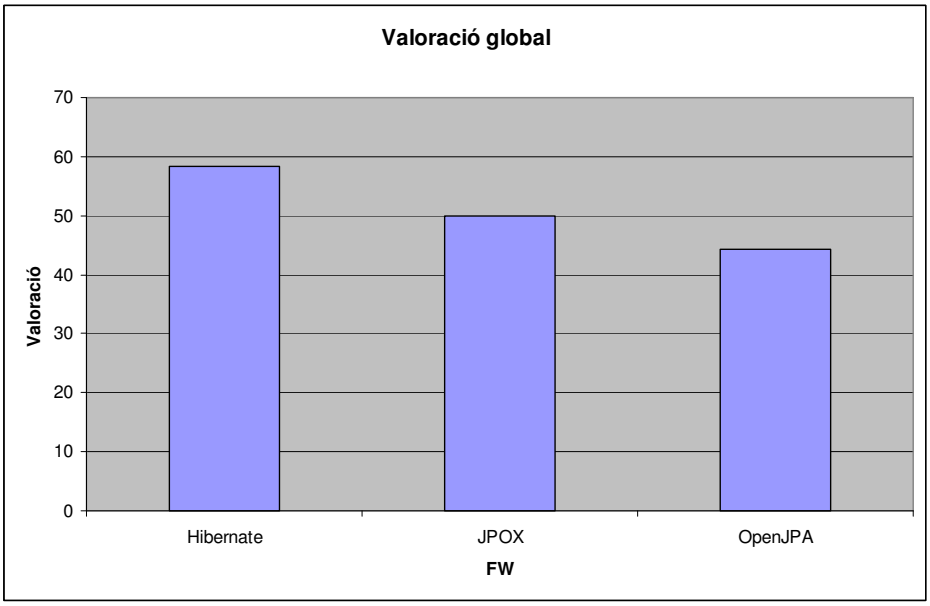
[1] No ha estat viable la generació del model a partir de OpenJPA. El primer problema detectat és el nombre de registres per transacció, i el segon problema que el temps és excessiu si es disminueix el nombre de registres per transacció.
[2] Inserció desordenada dins d'una transacció a diferència de JPOX i Hibernate (configuració per defecte, no vol dir que no es pugui sinó que en tot cas s'hauria de veure si es pot i especificar-ho).
[3] No soluciona automàticament els problemes de seqüències derivats de la migració de MySQL a un altre gestor de BBDD, JPOX ho té ben solucionat amb la taula d'objectes que crea en la primera execució.
[4] Problemes amb l'esquema de BBDD de PostgreSQL, apunta a la BBDD per defecte, esquema públic. Requereix de configuració addicional respecte els altres dos FW.
[5] Veure nota [3].
[6] Requereix de compilació especial que s'acaba traduint en plugins específics pels entorns en que es treballa.
[7] Hibernate ofereix tipus de bloqueigs que permeten decidir-lo a cada load de forma senzilla, a més incorpora tipus de bloqueigs addicionals a part dels estàndards que donen flexibilitat (tot i que cal usar-los amb coneixement per no perdre la portabilitat de l'aplicació).

Respecte a les proves d'especificació, tots tres FW demostren fer allò que diuen i tots realitzen tasques semblants (com per exemple definir anotacions addicionals a les que defineix l'estàndard).

| Especificació | Hibernate | JPOX | OpenJPA |
|---|---|---|---|
| espec_1 | 1 | 1 | 1 |
| espec_2 | 1 | 1 | 1 |
| espec_3 | 1 | 1 | 1 |
| **Resultat** | **3** | **3** | **3** |

Taula de resultats respecte les proves realitzades:

| | Hibernate | JPOX | OpenJPA |
|---|---|---|---|
| Proves automàtiques | 12 | 11 | 9 |
| Proves manuals | 6 | 4 | 4 |
| Especificació | 3 | 3 | 3 |
| **Resultat** | **21** | **18** | **16** |




Aquesta conclusió fa referència a les proves realitzades en aquest document. No pretén ser extensible a qualsevol tipus de prova ni determinar fins a quin punt un FW és més o menys recomanat per a utilitzar en un projecte.

## *5.2 Es pot determinar si un FW és millor que un altre?*

En els següents apartats es fa una reflexió sobre si el resultat de l'estudi es pot prendre de forma estricta com un resultat definitiu o falten paràmetres per determinar si un FW és millor que un altre.

Durant el transcurs del projecte es pot veure que hi ha paràmetres que tenen a veure amb la pròpia implementació, d'altres que tenen a veure amb l'especificació i d'altres amb la configuració del FW; però cal també pensar en altres factors que són determinants a l'hora de decidir quin és el FW que es vol utilitzar per a un projecte concret.

### 5.2.1 Factor de portabilitat

Durant el document s'ha parlat de la portabilitat entre Bases de Dades i entre servidors d'aplicacions; però, què passa si un dia decidim canviar de FW de persistència? Aleshores serà important que el que tenim fet amb un FW sigui fàcilment reutilitzable pel nou FW al que volem anar. Això s'aconsegueix mitjançant la utilització d'estàndards.

Hibernate ha sortit en primer lloc dins de les proves que s'han realitzat en el document, però en canvi no és un estàndard, amb la qual cosa a la llarga ens podria arribar a donar problemes.
Per altra banda, l'especificació que sembla que està prenent més força actualment (JPA) és la que pitjor ha sortit dins de les proves realitzades.

El que realment pot funcionar aleshores serà la implementació de JPA d'Hibernate, ja que ens hauria d'oferir el rendiment d'Hibernate que s'ha observat a les proves juntament amb la portabilitat de la pròpia especificació JPA (tenint en compte que introduïm limitacions, com per exemple les operacions en cascada).

### 5.2.2 Quanta gent utilitza el FW que estem estudiant?

Un altre paràmetre que no es té en compte en aquest projecte i que en els projectes que han de ser implementats en un entorn productiu pren molta força és: Quants projectes utilitzen aquest FW? Amb quina especificació?

Hi ha moltes decisions que escapen a la mesura tècnica pròpiament dita i que tenen en compte factors com la utilització, suport, documentació i d'altres paràmetres que tot i que a primera vista (des d'una òptica purament tècnica) poden no tenir molt de sentit, acaben desembocant en:

- Un projecte amb una implementació molt àmplia tindrà bona documentació, molts errors detectats i molts exemples en diferents entorns.
- Els servidors d'aplicacions acaben donant suport a aquelles implementacions majoritàries (incloent llibreries, publicant exemples de descriptors...).

## 5.3 Punts de millora del projecte

Hi ha diferents punts de millora que es podrien aplicar per a tenir un anàlisi més exhaustiu dels diferents FW utilitzats al projecte:

1) La capacitat dels FW d'operar en diferents servidors d'aplicacions J2EE.

He observat en el transcurs del projecte que hi ha alguns servidors d'aplicacions que donen suport natiu (inclouen les llibreries o personalitzacions de les mateixes) als diferents FW. Per tant és més senzill implementar en segons quins entorns amb un FW o un altre. En aquest projecte ha faltat temps per a poder realitzar un anàlisi (amb implementació) de la possibilitat d'inter-operació dels FW amb els diferents servidors d'aplicacions J2EE.

2) Analitzar els sistemes de caché que incorporen els FW.

Cada FW incorpora diferents sistemes de caché i permet utilitzar llibreries per a gestionar-lo. Aquesta funcionalitat pot augmentar o disminuir molt el rendiment d'un FW en quant a nombre de consultes que es llencen en una transacció. Per exemple, Hibernate manté caché a nivell de transacció, per tant la mateixa select dues vegades en una mateixa transacció, el primer cop accedirà a la Bases de Dades, mentre que el segon cop intentarà recuperar de caché (cal vigilar en utilitzar selects iguals, si per exemple utilitzem selects amb valors de timestamp, és probable que no funcioni correctament).

3) Suport per a eines de cerca.

Existeix algun FW que és capaç d'integrar-se amb eines de cerca com, per exemple, Hibernate que té una llibreria que permet indexar continguts d'Apache Lucene.

4) Determinar si els FW permeten a partir d'anotacions pròpies realitzar funcions addicionals.

Per exemple determinar si es poden o no realitzar funcions de validació en base a les anotacions .

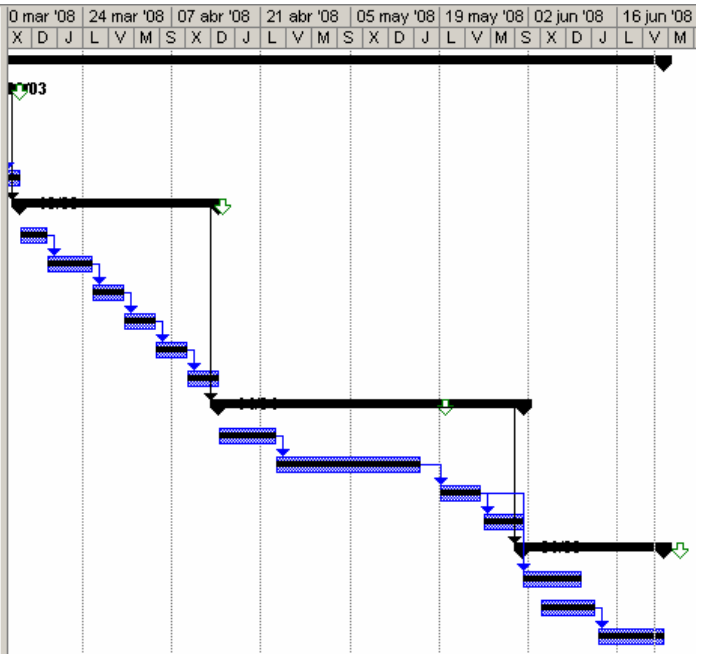
Alguna d'aquestes millores podrien considerar-se un projecte complert (per exemple fer un anàlisi exhaustiu de les eines de caché amb les seves diferents possibilitats, o implementar un sistema de cerca sobre Apache Lucene basat en Hibernate). D'altres, com per exemple la facilitat d'operar en diferents servidors J2EE, podrien considerar-se com extensió del propi projecte.

# 6. Seguiment de projecte.

El seguiment del projecte s'ha realitzat mitjançant l'eina de gestió de projectes Microsoft Project en base a la planificació inicial realitzada en la primera entrega.

La situació definitiva del pla de projecte es mostra a continuació amb les respectives modificacions en les tasques per tal de mostrar el % completat:

| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **PFC UOC** | **108,5 días** | **dom 09/03/08** | **lun 23/06/08** |
| **PAC1** | **5 días** | **dom 09/03/08** | **jue 13/03/08** |
| PAC 1 > Introducció | 1 día | dom 09/03/08 | dom 09/03/08 |
| PAC 1 > Definició objectius | 1 día | dom 09/03/08 | lun 10/03/08 |
| PAC 1 > Planificació de projecte | 3 días | lun 10/03/08 | jue 13/03/08 |
| **PAC2** | **32 días** | **jue 13/03/08** | **lun 14/04/08** |
| PAC2 > Definir capa estudi | 5 días | jue 13/03/08 | mar 18/03/08 |
| PAC2 > Seleccionar Frameworks a avaluar | 7 días | mar 18/03/08 | mar 25/03/08 |
| PAC2 > Definir proves automatitzades de mesura | 5 días | mar 25/03/08 | dom 30/03/08 |
| PAC2 > Definir mesures no automatitzades | 5 días | dom 30/03/08 | vie 04/04/08 |
| PAC2 > Definir mesures de compliment d'especificació del propi Framework | 5 días | vie 04/04/08 | mié 09/04/08 |
| PAC2 > Definir el entorn de treball a utilitzar | 5 días | mié 09/04/08 | lun 14/04/08 |
| **PAC3** | **49,22 días** | **lun 14/04/08** | **dom 01/06/08** |
| PAC3 > Configuració entorn de treball | 9 días | lun 14/04/08 | mié 23/04/08 |
| PAC3 > Codificació de les proves automatitzades | 22,98 días | mié 23/04/08 | jue 15/05/08 |
| PAC3 > Executar mesures no automatitzades | 7 días | lun 19/05/08 | dom 25/05/08 |
| PAC3 > Executar mesures de compliment d'especificació del propi Framework | 7 días | lun 26/05/08 | dom 01/06/08 |
| **Entrega Final** | **23,19 días** | **dom 01/06/08** | **lun 23/06/08** |
| Entrega final > Revissió i finalització del codi alpha | 10 días | dom 01/06/08 | mar 10/06/08 |
| Entrega final > Finalitzar documentació | 9 días | mié 04/06/08 | jue 12/06/08 |
| Entrega final > realització de presentació virtual | 11 días | vie 13/06/08 | lun 23/06/08 |




S'observa com per a poder arribar a la consecució del projecte ha estat necessari sobrecarregar els recursos en la última etapa, treballant en paral·lel en la tasca de revisió de codi i la documentació (memòria i presentació).