ESUELA SUPERIOR POLITECNICA DE CHIMBORAZO

FACULTAD DE INFORMATICA Y ELECTRONICA

ESCUELA DE INGENIERIA EN SISTEMAS

"ANÁLISIS HIBERNATE COMO TECNOLOGÍADE PERSISTENCIA DE OBJETOS SOBRE BASES DE DATOS RELACIONALES EN APLICACIONES EMPRESARIALES: CASO PRÁCTICO CONTROL DE BIENES DEL GOBIERNO MUNICIPAL DE CARLOS JULIO AROSEMENA TOLA."

TESIS DE GRADO

Previa a la obtención del título de

INGENIERO EN SISTEMAS INFORMATICOS

Presentado por:

OLJER ALFREDO CANDO CANDO

RIOBAMBA – ECUADOR

2013

AGRADECIMIENTO

En primer lugar a Dios, por estar junto a mí en cada paso, a mis queridos padres por su apoyo incondicional, sus consejos y ejemplo de superación.

A la Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica, en especial a la escuela de Ingeniería en Sistemas y sus distinguidos catedráticos.

Especial agradecimiento a la Ing. Natalia Layedra Director de Tesis, por sus sabios conocimientos, su don de gente y su confianza depositada en mi persona.

Oljer Alfredo Cando C.

DEDICATORIA

El presente trabajo lo dedico a mi hija Emily Paola que es la razón de mi vida y el tesoro más grande que Dios me regalo.

A mis padres Manuel Cando y Sebastiana Cando que se sacrificaron en post de mi bienestar, guiaron mis pasos con mucho amor, me enseñaron a continuar luchando para vencer los obstáculos, sin perder la esperanza de conseguir las metas trazadas, a pesar de los tropiezos y dificultades que se han presentado.

A mis Hermanas: María, Magdalena, Fabiola, Fani y Delia que son mi fortaleza y el pilar de apoyo para llegar a cumplir con mis objetivos.

A mi esposa Tania, por su respaldo, apoyo, paciencia y enseñarme que siempre hay una luz al final del camino.

A la Señora Norma Buenaño y Julio Aguirre por su apoyo en este trabajo.

Y de manera Especial a un ser maravilloso que siempre creyó en mí y que está conmigo apoyándome incondicionalmente y en todo momento y esa persona en mi querida madre Sebastiana Cando.

Oljer Alfredo Cando C.

FIRMAS DE RESPONSABILIDAD

NOMBRES	**FIRMAS**	**FECHA**
Ing. Iván Menes **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	----------	----------
Ing. Raúl Rosero **DIRECTOR DE LA ESCUELA DE INGENIERIA EN SISTEMAS**	----------	----------
Ing. Natalia Layedra **DIRECTOR DE TESIS**	----------	----------
Ing. Raúl Rosero **MIEMBRO DE TESIS**	----------	----------
Tlgo. Carlos Rodríguez **Dir. Dpto. CENTRO DOCUMENTACION**	----------	----------
NOTA DE LA TESIS	----------	

“Yo, Oljer Alfredo Cando Cando”, soy responsable de las ideas, doctrinas y resultados expuestos en esta tesis; y, el patrimonio intelectual de la tesis de Grado pertenece a la Escuela Superior Politécnica De Chimborazo”.

_____

Oljer Alfredo Cando Cando

ABREVIATURAS.

API: Application Programming Interface (Interfaz de Programación de Aplicaciones).

BBDD: Base de Datos.

BMP: Bean-Managed Persistence (Persistencia Gestionada por el Bean)

CMP: Container-ManagedPersistence (Persistencia Gestionada por el Contenedor)

CORBA: Common Object Request Broker Architecture.

CRUD: Crear, Eliminación, Actualización y Eliminación.

DAO: Data Access Object (Objeto de Acceso a Datos)

DDL: Data Definition Language (Lenguaje de definición de Datos).

DML: Data Manipulation Language (Lenguaje de Manipulación de Datos).

EJB: Enterprise Java Bean

GNU: GNU no es Unix.

HQL: Hibernate Query Languaje (Lenguaje de Consultas de Hibernate).

J2EE: Java Enterprise Edition

JCA: Arquitectura de Conexion Java

JDBC: Java Database Connectivity (Conector de Base de Datos Java).

JMX: Java Management Extensions (Administración de Extensiones Java)

JNDI: Java Naming and Directory Interface (Interfaz de Nombres y Directories Java)

JTA: Api para Transacciones Java.

MSF: Microsoft Solution Framework.

OODBMS: Object–Oriented Database Mangement System (Sistema de administracion de Base de Datos Orientada a Objetos).

ORM: Object Relational Mapping.

PAO: Programación Orientada a Aspectos

POJO: Plain Old Java Objtect.

POO: Programación Orientada a Objetos

QBE: Query By Example

RDBMS: Relational Database Management System (Sistema de Administracion de Base de Datos Relacional).

SGB: Sistema de Gestión de Base de Datos.

SGB: Sistema de Gestión de Bienes.

SQL: Structured Query Language (Lenguaje de Consulta Estructurados)

VO: Value Object.

XML: Extensible Markup Language (Lenguaje de Marcas Extensibles)

INDICE GENERAL

PORTADA
AGRADECIMIENTO
DEDICATORIA
ABREVIATURAS
INDICE GENERAL
INDICE DE TABLAS
INDICE DE FIGURAS
CAPÍTULO I .....
1. MARCO REFERENCIAL 17
1.1. Antecedentes.....17
1.2. Justificación 19
1.2.1. Justificación Teórica.....19
1.3. Objetivos 21
1.3.1. Objetivo General 21
1.3.2. Objetivos Específicos 21
CAPÍTULO II .....
2. PERSISTENCIA 22
2.1. ¿Qué es la persistencia? 22
2.1.1. Introducción.....22
2.1.2. ¿Qué es Persistencia de Objetos? 22
2.1.3. Métodos de Persistencia de Objetos 23
2.1.4. Base de Datos Orientadas a Objetos 25
2.1.5 Base de Datos Relacionales.....26
2.1.6 Impedancia Objeto-Relacional.....28
2.1.6 Arquitectura basada en capas 30
2.1.7. Persistencia de Objetos Manual 32
2.2 ORM.....33
2.2.1 ¿Qué es ORM? 34
2.2.2. Componentes de un ORM 34
2.2.3 El ORM se puede usar de diferentes maneras. 38
2.2.4. Ventajas de utilizar un ORM 40
2.2.5. Desventajas de utilizar un ORM 41
CAPíTULO III .....
3. FRAMEWORKHIBERNATE 42

3.1 Introducción.....42
3.2 Historia de Hibernate.....43
3.3 Que es Hibernate.....43
3.4 ¿Por qué el uso de este Framework?.....44
3.5 Objetivos de Hibernate.....47
3.6 Características de Hibernate.....48
3.7 Arquitectura.....48
3.7.1 Estados de instancia.....52
3.7.2 Integración JMX.....53
3.7.3 Soporte JCA.....54
3.7.4 Sesiones contextuales.....54
3.8.2 Ejemplo simple de POJO.....58
3.8.3 Implementacion un constructor sin argumentos.....59
3.8.4 Declaracion de métodos de acceso y de modificación para los campos persistentes (opcional).....60
3.8.5 Implementación de herencia.....60
3.8.6 Modelos dinámicos.....60
3.9.1 Generación automática de esquemas.....64
3.10 Mapeo O/R Básico.....69
3.10.1 Declaración de mapeo.....69
3.10.2. Mapeo de Hibernate.....71
3.10.3 Clase.....72
3.10.4. Id.....77
3.10.7. Discriminador.....84
3.10.8. Versión (opcional).....85
3.10.9. Timestamp (opcional).....87
3.10.10 Property.....88
3.10.11. Relacion Many-to-one.....91
3.10.12. One-to-one.....92
3.10.13. Componente y componente dinámico.....93
3.10.14. Subclase.....94
3.10.15. Join.....94
3.11 TIPOS DE DATOS HIBERNATE.....96
3.11.1. Entidades y Valores.....96
3.11.2. Tipos de valores básicos.....98
3.11.3 Tipos de valor personalizados.....99

3.12. Mapeo de una clase más de una vez 100
3.13. Identificadores SQL en comillas 100
3.14. Alternativas de metadatos 101
3.14.1. Utlización de Anotaciones JDK 5.0 101
3.15. Propiedades generadas 102
3.16. Transacciones y concurrencias 102
3.16.1. Ámbitos de sesión y de transacción 103
3.16.2 Conversaciones largas 105
3.17. Manejo de excepciones 107
3.18. Tiempo de espera de la transacción 108
3.19. Control de concurrencia optimista 110
3.20. Modos de liberación de la conexión 110
3.21. TIPOS DE CONSULTAS QUE SOPORTA HIBERNATE 112
3.21.1. HQL 112
3.21.2 SQL 126
3.21.3 QBC 139
3.22.4 QBE 147
CAPíTULO IV .....
4. ANALISIS COMPARATIVO 153
4.1 Introduccion 153
4.2 Determinacion de las herramientas a comparar 154
4.2.1 Hibernate 154
4.2.2 EclipseLink 155
4.3. ANALISIS DE LAS HERRAMIENTAS 156
4.3.1. Hibernate 156
4.3.2 Eclipselink 162
4.4. Determinación de los parámetros de comparación. 168
4.4.1. Parámetro 1: Mapeo Objeto Relacional 168
4.4.2. Parámetro 2: Producto 168
4.4.3. Parámetro 3: Mecanismo 169
4.4.4. Parámetro 4: Comportamiento 169
4.5. Determinación de las variables para los parámetros de Comparación 169
4.6. Análisis comparativo 170
4.6.1. Parámetro 1: Mapeo Objeto-relacional 172
4.6.2. Parámetro 2: Producto 178
4.6.3. Parámetro 3: Rendimiento 182

4.6.4. Parámetro 4: Comportamiento 187
4.7 Análisis de los resultados 191
4.8 Demostración de la hipótesis.....192
4.8.1. Modelo para la comprobación de la hipótesis 193
4.8.2. Planteamiento de la hipótesis 194
4.8.3. Selección de nivel de significancia. 194
4.8.4. Descripción de la muestra. 194
4.8.5. Especificación del estadístico 194
4.8.6. Especificación de las regiones de rechazo y aceptación. 195
4.8.7. Recolección de datos y cálculos de los estadísticos 195
4.8.8. Tabulación de la información 195
4.8.9 Decisión estadística.....208
CAPíTULO V.....
5. DISEÑO Y DESARROLLO DE LA APLICACIÓN 209
Introducción.....209
5.1 Microsoft Solution Framework 209
5.1.1 Definición 209
5.1.2 Fases 209
5.1.3 Ventajas 210
5.1.4 Desventajas 211
5.1.5 Fase de Visión.....211
5.1.6 Definición del Problema.....211
5.1.7 Visión del Proyecto.....212
5.2 Perfiles de Usuario 213
5.3 Ámbito del Proyecto 214
5.3.1 Requerimientos funcionales 215
5.3.2 Requerimientos no funcionales 217
5.4 Objetivo del Proyecto 217
5.4.1 Objetivos del Negocio.....217
5.4.2 Objetivos de Diseño 218
5.5 Riesgos 218
5.6 Identificación del Riesgo.....219
5.6.1 Análisis de Riesgos 220
5.6.2 Planeación y programación del Riesgo.....223
5.7 Planificación Inicial 233
5.7.1 Factibilidad 233

7.8 Fase de Planificación 237
5.8.1 Diseño Conceptual 237
CONCLUSIONES
RECOMENDACIONES
GLOSARIO
BIBLIOGRAFÍA
ANEXOS

INDICE DE TABLAS

Tabla III.I. Resumen de atributos....66
Tabla III.II. Opciones de Línea de Comandos de SchemaExport....68
Tabla III.III. Propiedades de Conexión del Shema Export....69
Tabla III.IV. Nombres con inyección alias....131
Tabla III.V. Ejemplo de QBE....148
Tabla IV.VI. Tabla de Variables e indicadores a comparar....169
Tabla IV.VII. Escala de calificación para parámetros de comparacion....170
Tabla IV.VIII. Escala de valoración Cuantitativa....171
Tabla IV.IX. Significado de nomenclatura....171
Tabla IV.X. Significado de variables....172
Tabla IV.XI. Resumen variables parámetro Mapeo objeto-relacional....173
Tabla IV.XII. DBMs Soportados por Hibernate....175
Tabla IV.XIII. DBMs Soportados por Eclipselink....176
Tabla IV.XIV. Resumen variables parámetro Producto....180
Tabla IV.XV. Escenario 1. Hibernate....183
Tabla IV.XVI. Escenario 2. Eclipselink....184
Tabla IV.XVII. Resumen variables parámetro Rendimiento....186
Tabla IV.XVIII. Resumen variables parámetro Comportamiento....189
Tabla IV.XIX. Definición de Variables....193
Tabla IV.XX. Definición de Indicadores....193
Tabla IV.XXI. Valoración....195
Tabla IV.XXII. Resultados Pregunta 1 sin ningún sistema....196
Tabla IV.XXIII. Resultados Pregunta 2 sin ningún sistema....196
Tabla IV.XXIV. Resultados Pregunte 3 sin ningún sistema....197
Tabla IV.XXV. Resultados Pregunta 4 sin ningún sistema....197
Tabla IV.XXVI. Resultados Pregunta 5 sin ningún sistema....198
Tabla IV.XXVII. Resultados Pregunta 6 sin ningún sistema....198
Tabla IV.XXVIII. Resultados Pregunta 7 sin ningún sistema....198
Tabla IV.XXIX. Resultados Pregunta 8 sin ningún sistema....199
Tabla IV.XXX. Resultados pregunta 1 para el sistema SGB....200
Tabla IV.XXXI. Resultados Pregunta 2 para el sistema SGB....200
Tabla IV.XXXII. Resultados Pregunta 3 para el sistema SGB....201
Tabla IV.XXXIII. Resultados Pregunta 4 para el sistema SGB....201
Tabla IV.XXXIV. Resultados Pregunta 5 para el sistema SGB....202
Tabla IV.XXXV. Resultados Pregunta 6 para el sistema SGB....202
Tabla IV.XXXVI. Resultados Pregunta 7 para el sistema SGB....202
Tabla IV.XXXVII. Resultados Pregunta 8 para el sistema SGB....203
Tabla IV.XXXVIII. Resultado final....204
Tabla IV.XXXIX. Cálculos Estadisticos....207
Tabla V.XL. Perfiles de Usuario....214
Tabla V.XLI. Identificación de Riesgos....219
Tabla V.XLII. Valoración de riesgos....220
Tabla V.XLIII. Probabilidad....220
Tabla V.XLIV. Impacto del Riesgo....221
Tabla V.XLV. Riesgo-Impacto....221
Tabla V.XLVI. Impacto-Probabilidad....221
Tabla V.XLVII. Tabla tolal Riesgo....222

Tabla V.XLVIII. Pioridades del riesgo 222
Tabla V.XLIX. Gestión del Riesgo 1 223
Tabla V.L. Gestión del Riesgo 2 224
Tabla V.LI. Gestión del Riesgo 3 226
Tabla V.LII. Gestión del Riesgo 4 227
Tabla V.LIII. Gestión del Riesgo 5 228
Tabla V.LIV. Gestión del Riesgo 6 230
Tabla V.LV. Gestión del Riesgo 7 231
Tabla V.LVI. Hardware existente 233
Tabla V.LVII. Hardware requerido 233
Tabla V.LVIII. Software existente 234
Tabla V.LIX. Software requerido 234
Tabla V.LX. Recurso Humano Requerido 234
Tabla V.LXI. Personal a Capacitar 235
Tabla V.LXII. Personas Alternativas para Super Administrador 272
Tabla V.LXIII. Personas alternativas para Administradores 273
Tabla V.LXIV. Personas alternativas para usuarios tipo invitado 273
Tabla V.LXV. Glosario de Terminos 274
Tabla V.LXVI. Autenticacion de usuarios 275
Tabla V.LXVII. Usuario Administrador 277
Tabla V.LXVIII. Usuario invitado 279

INDICE DE FIGURAS

Figura II.1.Proceso de serialización....24
Figura II.2. Almacenamiento de Objetos vs mapeo de Objetos 25
Figura II.3. Impedancia Objeto Relacional....28
Figura II.4. Capa de Persistencia es la base de una Arquitectura en capas 30
Figura II.5. Acceso a datos mediante JDBC 32
Figura II.6. Conversión de datos....34
Figura II.7. Interacción de Componentes....36
Figura III.8. Arquitectura Base de Hibernate....44
Figura III.9. Uso del FrameworkHibernate 44
Figura III.10. Arquitectura de alto nivel de Hibernate....48
Figura III.11. Arquitectura mínima de Hibernate 49
Figura III.12. Arquitectura completa de Hibernate 50
Figura III.13. Ciclo de vida de persistencia de Hibernate.....56
Figura IV.14. Arquitectura de Hibernate 158
Figura IV.15. Arquitectura de Eclipselink....163
Figura IV.16. Componentes de Eclipselink....164
Figura IV.17. Comparacion estadística del parámetro Mapeo Objeto Relacional 178
Figura IV.18. Comparacion estadística del parámetro Producto....182
Figura IV.19. Test de Rendimiento de Eclipselink 185
Figura IV.20. Test de Rendimiento de Hibernate....185
Figura IV.21. Comparacion estadística del parámetro Rendimiento 187
Figura IV.22. Comparacion estadística del parámetro Comportamiento....190
Figura IV.23. Resultado Final por parámetro....191
Figura IV.24. Diagrama general de resultados 192
Figura IV.25. Regiones de rechazo y aceptación 195
Figura IV.26. Resultado encuesta situación actual. 199
Figura IV.27. Resultado encuesta Sistema SGB 203
Figura IV.28. Resultado final 205
Figura IV.29. Zona de la Prueba estadistica....208
Figura V.30. Fases de MSF....210
Figura V.31. Caso de uso Súper Administrador....269
Figura V.32. CU Administrador Departamento 270
Figura V.33. CASO USO ADMINISTRADOR (BODEGA)....270
Figura V.34. Caso de Uso Administrador (Combustible) 271
Figura V.35. Casos de Uso Administrador (Mantenimiento) 271
Figura V.36. Casos de Uso Invitado....272
Figura V.37. Diagrama de Secuencia de Autenticacion de usuarios....281
Figura V.38. Diagrama de secuencias para la creación de un Departamento....282
Figura V.39. Diagrama de Secuencia para dar de baja un equipo....282
Figura V.40. Diagrama de Clase 283

CAPÍTULO I

1. MARCO REFERENCIAL

1.1. ANTECEDENTES

En la actualidad diferentes capas de las aplicaciones modernas se construyen utilizando la programación orientada objetos para implementar la lógica de negocio y el modelo de base de datos relacional para almacenamiento de datos.

El modelo relacional para gestión de base de datos se basa en la lógica de predicados y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Sin embargo, el paradigma de programación orientado aobjetos o POO usa objetos en sus interacciones para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, como encapsulamiento, polimorfismo, acoplamiento, herencia, cohesión, abstracción y trata con objetos, atributos, métodos y relaciones entre objetos.

Cuando se quiere hacer que los objetos sean persistentes utilizando para ello una base de datos relacional, uno se da cuenta de que hay una contrariedad entre estos dos paradigmas, la también llamada diferencia objeto-relacional (“object-relational-

map”). Una técnica de programación que nos permitamanteneerestos objetos persistentes con los dos modelos son los ORM.

El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relationalmapping ORM) es una técnica de programación para convertir datos entre el sistema de tipos utilizados en un lenguaje de programación orientada a objetos y el utilizado en la base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos.

Hibernate es un ORM para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones y ha conseguido en un tiempo record una excelente reputación en la comunidad de desarrollo posicionándose claramente como el producto OpenSource líder en este campo gracias a sus prestaciones, buena documentación y estabilidad.

Es así que se presenta Hibernate, como un Framework orientado a la persistencia.

¿**Qué es la persistencia?**

Proviene del Latín ***persistere*** que significa durar por largo tiempo, en el ámbito informático la mayoría de los programas informáticos actuales necesitan preservar los datos para su posterior uso, esto es más frecuente el uso de base de datos relacionales.

Hibernate parte de una filosofía de mapear objetos Java "normales", también conocidos en la comunidad como "POJO's5" (Plain Old Java Objects). No contempla la posibilidad de automatizar directamente la persistencia de EntityBeans tipo BMP (es decir, generar automáticamente este tipo de objetos), aunque aun así es posible combinar Hibernate con este tipo de beans utilizando los patrones para la delegación de persistencia en POJO's.

Hibernatepermite al desarrollador ser más productivo, reduciendo costos y aumentando ganancias. Ya que existen estudios que demuestran que el 35% del

código de una aplicación se produce como consecuencia del mapeado (correspondencia) entre los datos de la aplicación y el almacén de datos.

Con la herramienta seleccionada se pretende desarrollar la solución propuesta para el Gobierno Autónomo descentralizado de Arosemena Tola que en la actualidad no cuenta con un sistema que le permita controlar el uso de los materiales adquiridos para una obra ya que se despacha sin control de acuerdo a la existencia en bodega, lo que causa un déficit de materiales que fueron específicos para una obra.

Otro problema existente es en el control de combustible que se despacha por solicitud del chofer, el problema surge ya que son pocos los kilómetros recorridos en relación a la cantidad de combustible cargado.

1.2. JUSTIFICACIÓN

1.2.1. JUSTIFICACIÓN TEÓRICA

El modelo relacional para gestión de base de datos se basa en la lógica de predicados y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Su Idea fundamental es el uso de relaciones las mismas que se pueden considerar en forma lógica como conjunto de datos llamadas tuplas. Sin embargo, el paradigma de programación orientado a objetos oPOO usa objetos en sus interacciones para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, como encapsulamiento, polimorfismo, acoplamiento, herencia, cohesión, abstracción y trata con objetos, atributos, métodos y relaciones entre objetos.

Cuando se quiere hacer que los objetos sean persistentes utilizando para ello una base de datos relacional, uno se da cuenta de que hay una contrariedad entre estos dos paradigmas, la también llamada diferencia objeto-relacional (“object - relational -

map”). Una técnica de programación que nos permita mantener estos objetos persistentes con los dos modelos son los ORM.

Hibernate es Tecnología de JBoss para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Hibernate no requiere de esquemas especiales para la construcción de los objetos, por lo cual es conocido como basado en POJO (Plain Old JavaObjects). Adicionalmente, permite realizar consultas sobre la base de datos usando SQL tradicional o HQL, un lenguaje orientado al manejo de objetos

Justificación Aplicativa

La razón principal para el desarrollo de este proyecto nace del resultado de la evaluación y posterior reflexión de las necesidades vistas durante los procesos deControl de uso de los bienes frente al rendimiento de los mismos en el Gobierno AutónomoMunicipal de Arosemena Tola, por lo que es posible concebir una solución informática aplicando latecnología Hibernate.

La solución contara con los siguientes Módulos:

- Módulo de autenticación de usuarios administradores y clientes definidos por la institución, basado en roles, lo que permitirá desplegar los diferentes procesos descritos en los siguientes puntos.
- Módulo de Control de materiales y suministro de oficina por departamentos.

  Este módulo tiene como objetivo llevar un control adecuado del uso de los materiales solicitados por los departamentos

- Módulo de Control consumo y rendimiento de combustible en la institución.

Permitirá controlar y transparentar el uso del combustible, conocer la eficiencia y rendimiento de las maquinarias y vehículos.

- Módulo de Control de Materiales de Construcción por Obras.

  Permitirá Controlar los materiales adquiridos para la una obra determinada y a la vez conocer la cantidad y calidad de los materiales para las auditorias que se realizan.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Analizar la tecnología Hibernate, como Framework de persistencia de aplicaciones empresariales, y aplicar al desarrollo del sistema de control de bienes del Gobierno Municipal de Carlos Julio Arosemena Tola.

1.3.2. OBJETIVOS ESPECÍFICOS

1. Analizar diversas herramientas de persistencia de Java basabas en ORM que se puedan implementar en las empresas para solución de problemas.
2. Estudiar las características, ventajas y desventajas de latecnología Hibernate.
3. Desarrollar la solución propuesta, usando Hibernate como Framework de persistencia.
4. Evaluar los resultados obtenidos mediante el uso de dicho Framework.

1.4. Hipótesis

El desarrollo de una aplicación web, basada en el Framework Hibernate como apoyo al nivel Administrativo mejorará el control y uso de los bienes internos y externos del Gobierno Municipal.

CAPÍTULO II

2. PERSISTENCIA

2.1. ¿QUÉ ES LA PERSISTENCIA?

2.1.1. INTRODUCCIÓN

Cuando iniciamos el desarrollo de una aplicación específicamente una orientada a objetos, uno de los primeros requerimientos que debemos resolver es la integración con una base de datos relacional para guardar, actualizar y recuperar la información que utiliza la aplicación. En este capítulo se presenta los fundamentos, problema y la alternativa de solución común para la persistencia de Objetos en base de datos relacionales.

2.1.2. ¿QUÉ ES PERSISTENCIA DE OBJETOS?

Podemos encontrar diferentes definiciones del término persistencia, según distintos puntos de vista y autores. Veamos dos que con más claridad y sencillez, concretan el concepto de persistencia de objetos. La primera definición dice así: “La persistencia de objetos significa que los objetos individuales pueden sobrevivir al proceso de la

aplicación; pueden ser guardados a un almacén de datos y ser reconstruidos más tarde [1].”

La otra definición dice: Se llama persistencia de objetos a su capacidad para guardarse y recuperarse desde un medio de almacenamiento.

En definitiva la persistencia de objetos es la capacidad que tienen los objetos desobrevivir al proceso que los creó; permitiendo al programador almacenar, transferir, yrecuperar el estado de los objetos.

2.1.3.MÉTODOS DE PERSISTENCIA DE OBJETOS

En la actualidad podemos identificar tres formas usuales de persistir objetos: [LIB02]

- Serialización.
- Bases de Datos Orientadas a Objetos (ODBMS).
- Bases de Datos Relacionales.

2.1.3.1 Serialización

La serialización es el proceso de convertir un objeto en una secuencia de bytes para conservarlo en memoria, una base de datos o un archivo. Su propósito principal es guardar el estado de un objeto para poder crearlo de nuevo cuando se necesita. El proceso inverso se denomina deserialización.

[1]http://repositorio.utn.edu.ec/bitstream/123456789/571/1/Tesis.pdf

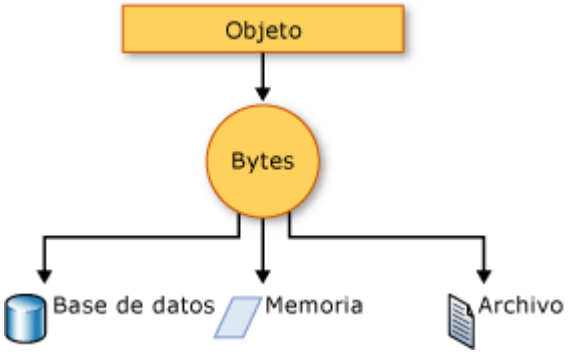


Figura II.1.Proceso de serializacion

Fuente: http://msdn.microsoft.com/es-es/library/ms233836%28v=vs.80%29.aspx

El objeto se serializa en una secuencia que, además de los datos, contiene información sobre el tipo de objeto, como la versión, referencia cultural y nombre de ensamblado. Esa secuencia se puede almacenar en una base de datos, un archivo o en memoria. La serialización permite al desarrollador guardar el estado de un objeto y volver a crearlo cuando es necesario, y proporcionar almacenamiento de objetos e intercambio de datos. A través de la serialización, un desarrollador puede realizar acciones como enviar un objeto a una aplicación remota por medio de un servicio Web, pasar un objeto de un dominio a otro, pasar un objeto a través de un firewall como una cadena XML o mantener la seguridad o información específica del usuario entre aplicaciones.

Sin embargo no soporta transacciones, consultas o acceso compartido a los datos entre usuarios múltiples y se la utiliza sólo para proporcionar persistencia en aplicaciones simples o en entornos empotrados que no pueden gestionar una base de datos de forma eficiente.

2.1.4. BASE DE DATOS ORIENTADAS A OBJETOS

Un ODBMS, o base de datos orientada a objetos, proporciona un método transparente para la persistencia. Permite consultar y trabajar con objetos directamente [2].

La persistencia transparente se refiere a la habilidad de manipular directamente, sin traducción o conversión, datos almacenados en la base utilizando un entorno de programación basado en objetos como Smalltalk, Java o C#. Esto contrasta con los RDBMSs donde se utiliza un sublenguaje como SQL o una interfaz de operación como ODBC o JDBC, luego, se utiliza código adicional para hacer la conversión a objetos.

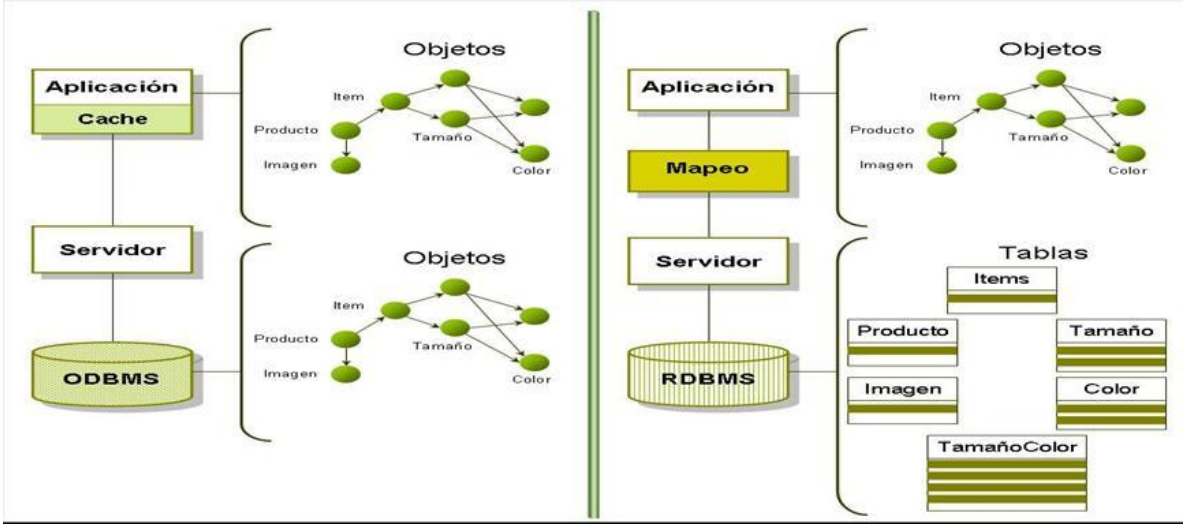


Figura II.2.Almacenamiento de Objetos vs mapeo de Objetos

Fuente: http://kuainasi.ciens.ucv.ve/db4o/DB4o-P1.htm

La mayoría de los ODBMSs implementan un esquema de persistencia por capacidad de alcance. Ello significa que cualquier objeto referenciado por un objeto persistente también es persistido. Normalmente el programador puede especificar la profundidad de operación de este esquema en un árbol de objetos. De esta manera, conjuntos completos de objetos pueden ser almacenados y recuperados con una sola llamada; el

[2]http://www.cpe.ku.ac.th/~plw/oop/e_book/Hibernate_in_action.pdf

ODBMS automáticamente los detalles de mantenimiento de las referencias cuando se guardan y recuperan objetos[3].

Las bases de datos orientadas a objetos son quizás la forma más sencilla de persistir un modelo de objetos, aunque el mercado de las tecnologías de bases de datos orientadas a objetos es aún pequeño e inestable comparado con el mercado de las bases de datos relacionales.

2.1.5BASE DE DATOS RELACIONALES

2.1.5.1 Modelo Relacional

Una descripción muy sencilla y directa del modelo relacional, es aquella en donde se define que la responsabilidad de las bases de datos relacionales es modelar la información basándose en relaciones definidas en conjuntos finitos de valores llamados dominios.

Una relación es un conjunto de listas ordenadas de valores llamadas tuplas. Cada tupla es un elemento del producto cartesiano de dominios. Cada ocurrencia de un dominio en la definición de una relación se denomina atributo, y el mismo dominio puede llegar a repetirse dentro de ella. Los dominios y las relaciones son implementados como tablas con m filas y n columnas. Cada fila corresponde a una tupla y cada columna a un atributorelacional. Por eso es que a lo largo de esta tesis se usará el concepto de tabla, columnay fila en lugar de relación, atributo y tupla.

Cada columna de una tabla tiene un tipo y un nombre para poder ser referenciada sinimportar su posición relativa. El tipo de una columna se limita a un conjunto pequeño detipos predefinidos como integer, varchar o date. Las tablas se definen usualmente conrestricciones impuestas a sus filas para evitar la duplicación de datos o

[3]http://www.db4o.com/espanol/db4o%20Whitepaper%20%20Bases%20de%20Objetos.pdf

dependenciascruzadas. Una restricción sobre una tabla es requerir que cada fila sea identificableunívocamente a partir de un subconjunto de sus columnas. A este subconjunto se lodenomina llave primaria. Cuando esta llave esta compuesta por una única columna se ladenomina llave simple sino en caso de contar con más de una columna se dice que lallave es compuesta.

El acceso a los datos se realiza a través de un conjunto básico de operaciones: selección,proyección, producto, join, unión, intersección y diferencia. Estas operaciones se expresan a través de un lenguaje denominado SQL (Lenguaje de Consultas Estructurado) usado para guardar, recibir y modificar información en la base de datos. Para recibir información se debe realizar una selección (caracterizada por la consulta SELECT en SQL). La selección de las filas permite especificar ciertas condiciones para los atributos acompañada generalmente con una proyección que significa que sólo un subconjunto de las columnas son seleccionadas. Para consultas más complejas se utiliza el operador JOIN que crea una tabla temporal con conjuntos de columnas pertenecientes a dos o más tablas. El resultado de una selección es generalmente un conjunto de filas o RecordSet. Para recorrer este conjunto los lenguajes de programación brindan un cursor y operaciones de navegación encapsuladas en librerías.

De las tres formas de persistencia, sólo las bases de datos relacionales han demostrado ser escalables, robustas y lo suficientemente estándares para las aplicaciones empresariales. No obstante cuando se quiere persistir los objetos utilizando una de ellas, se puede observar que hay un problema de compatibilidad entre el paradigma de la Orientación a Objetos y el modelo relacional, la también llamada diferencia o impedancia objeto-relacional

2.1.6 IMPEDANCIA OBJETO-RELACIONAL

"Impedancia Objeto-Relacional", se define como un conjunto de dificultades técnicas que surgen cuando una base de datos relacional se usa en conjunto con un programa escrito bajo el paradigma de la Orientación a Objetos[4].

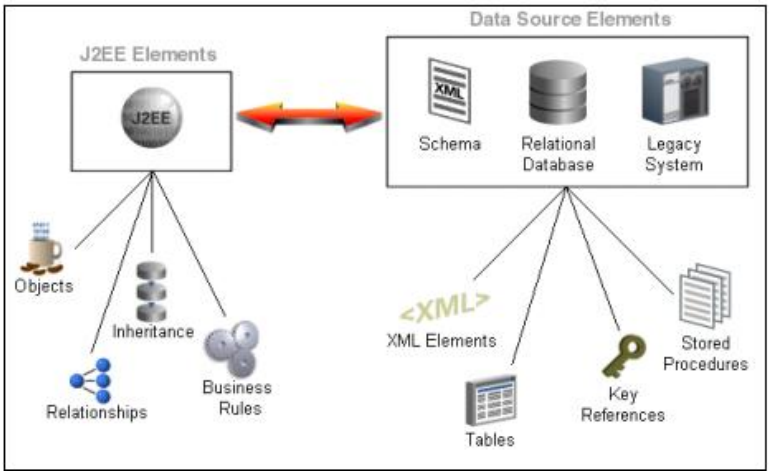


Figura II.3.Impedancia Objeto Relacional

Fuente: http://sqltech.cl/doc/oas10gR31/web.1013/b28218/undtl.htm

Un ejemplo claro de esta impedancia se observa en el hecho que en el mundo de la programación orientada a objetos, se tiene un claro sentido de la pertenencia, a cada objeto le pertenecen sus correspondientes atributos; por ejemplo para el objeto Agenda Telefónica podríamos especificar como atributos a una colección de objetos llamados "persona", en la que a cada persona le corresponde su correspondiente atributo "teléfono", al transformar esto hacia el mundo relacional se ocuparía más de una tabla para almacenar la información, este simple hecho, hace notar que las tablas del modelo relacional son inconscientes de cómo están relacionadas con otras tablas a un nivel fundamental, puesto que aún cuando posean constraints para definir sus

[4] Richard S. Java Persistence for Relational Databases. New-York-EE.UU, Apress, © 2003.

relaciones, para reconstruir el objeto originalmente persistido se debe construir un query, y dicho query debe especificar explícitamente como se relacionan las tablas entre sí, con esto se demuestra además que el lenguaje SQL a pesar de los constraints se mantiene inconsciente de las relaciones que a nivel de objeto poseen las tablas entre ellas.

Así como lo anteriormente expuesto se pueden enumerar distintos problemas que surgen entre los dos modelos:

Reglas de Acceso: En el modelo relacional los atributos pueden ser accesados y/o modificados a través de operadores relacionales predefinidos, mientras que en el modelo orientado a objetos, se permite que cada clase defina la forma en que serán alterados los atributos así como la interfase que ocupará para ello.

Ataduras del Esquema: Los objetos del modelo de la POO, no deben seguir ningúnesquema en cuanto a que atributos deben o pueden tener, puesto que son definidos porel programador, mientras que las tablas deben seguir el esquema entidad-relación.

Identificador único: Las llaves primarias de una fila tienen generalmente una forma depoder representarse como texto visible, mientras que los objetos no requieren unidentificador único externamente visible.

Estructura vs Comportamiento: La orientación a objetos se concentraprimordialmente en asegurar que la estructura del programa sea razonable (entendible,extensible, reusable, segura, etc), mientras que los sistemas relacionales ponen el énfasisen tipo de comportamiento que el sistema tendrá una vez en producción (eficiencia,adaptabilidad, rapidez, etc.). Los métodos de la POO asumen que el principal usuario delcódigo orientado a objetos y sus beneficios es el desarrollador de aplicaciones, mientrasque el modelo relacional enfatiza que la forma en que los usuarios finales perciben elcomportamiento del sistema es mucho más importante.De todo esto surge la necesidad de utilizar algún mecanismo para integrar

la informacióncontenida en nuestros objetos con los datos almacenados en la base de datos relacional.

Esto típicamente se logra a través de una capa de traducción objeto – relacional.A continuación se explica la arquitectura de una aplicación y una de las alternativas máscomunes para la transformación de objetos a bases de datos relacionales.

2.1.6 ARQUITECTURA BASADA EN CAPAS

Para organizar una aplicación empresarial, la industria del software ha convergido en unaArquitectura basada en Capas, dividiendo el sistema en tres capas básicas: la capa depresentación, la capa de lógica de dominio y la capa de persistencia, como se muestra enla siguiente figura.

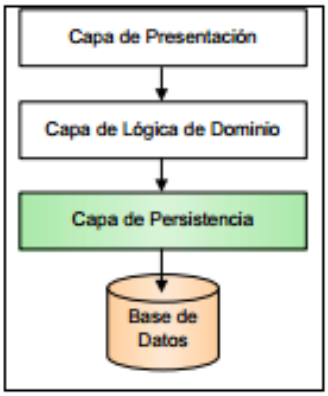


Figura II.4. Capa de Persistencia es la base de una Arquitectura en capas

Fuente: http://salvabasile.com.ar/linq-parte-1/

El principio detrás de esta arquitectura es que cada capa dependa sólo de los elementoscontenidos en ella o en las capas situadas por debajo, teniendo responsabilidades biendefinidas y evitando cualquier tipo de acoplamiento con las capas superiores.

Capa de Presentación

Maneja la interacción entre el usuario y la aplicación. A veces el nombre presentación presta a pensar en sólo salida de la aplicación, pero en realidad maneja la interacción en ambas direcciones. En algunas ocasiones el usuario puede ser otro sistema comunicándose por ejemplo a través de un servicio remoto.

Esta comunicación se hace especialmente notoria en ambientes Web, donde la capa de presentación no sólo tiene que crear documentos entendibles por los usuarios sino manejar los mensajes enviados por el browser como consultas o datos de formularios.

Para esto se suele utilizar un esquema de Controladores y Vistas, donde los controladores son el eje entre las capas inferiores y las vistas, el patrón de diseño Model View Controller es un ejemplo popular de esta forma de estructurar la aplicación. Algunos autores dividen los controladores en una capa separada llamada Capa de Aplicación.

Capa de Lógica de Dominio

Representa conceptos de negocio como reglas o estados. Lo que distingue a esta lógica por sobre el resto de la aplicación es que mantiene los conceptos centrales del proceso, siendo muchas veces la ventaja competitiva por sobre el resto de los productos. Es por ello que, a pesar de que por lo general sólo constituye un conjunto de módulos pequeños comparado el resto de la capas, suele tener mayores requerimientos como tests automáticos o revisiones. Esta importancia se realiza en las aplicaciones empresariales, la lógica irregular y propensa a cambios requiere un tratamiento especial.

Capa de Persistencia

Esta capa brinda servicios para sincronizar la capa de lógica con un medio de almacenamiento. Para esto se deben identificar los objetos en memoria que deben sobrevivir a la ejecución del programa teniendo así una persistencia de largo plazo.

Dependiendo del tipo de aplicación existen distintos requisitos para esta capa. Por ejemplo en una aplicación de diseño CAD generalmente cuando se edita un

documento se trae un gran conjunto de información a memoria desde el medio de almacenamiento ya que traer subconjuntos de información haría que la aplicación tenga un tiempo de respuesta pobre. En cambio en una aplicación empresarial generalmente las operaciones se concentran sólo en un grafo limitado de objetos, por lo que traer toda la información disponible es un gasto inaceptable. Para esto debe existir un especial interés en la optimización entre la cantidad de información que se trae a memoria y la realmente utilizada.

2.1.7. PERSISTENCIA DE OBJETOS MANUAL

Java Database Connectivity (JDBC)

Java DataBase Connectivity es el API de Java que define cómo una aplicación accederá auna base de datos, independientemente del motor de base de datos. Se lo realizamediante ejecuciones de sentencias SQL.

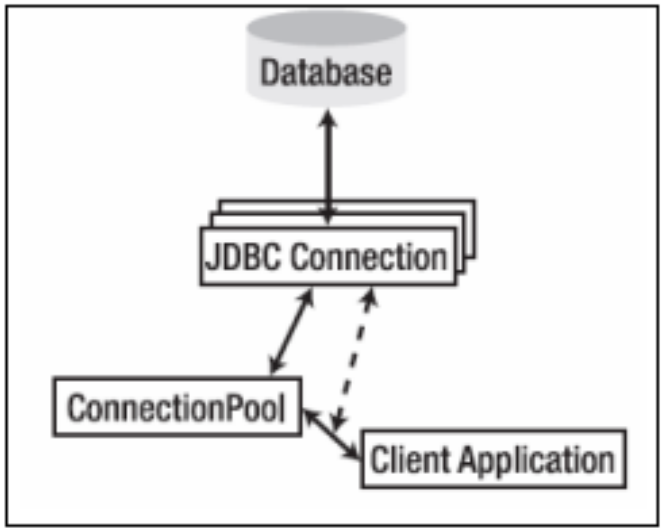


Figura II.5. Acceso a datos mediante JDBC

Fuente: Autor

No obstante, es necesario gestionar explícitamente los valores de los campos y su proyección en tablas de una base de datos relacional.

Por tanto:

- Hay que tratar con dos modelos de datos, lenguajes y paradigmas de acceso a los datos, muy diferentes (Java y SQL).
- El esfuerzo necesario para implementar el mapping entre el modelo relacional y el modelo de objetos es demasiado grande:
    - Muchos desarrolladores nunca llegan a definir un modelo de objetos para sus datos.
    - Se limitan a escribir código Java procedural para manipular las tablas de la base de datos relacional subyacente.
    - Se pierden los beneficios y ventajas del desarrollo orientado a objetos.

Esta forma de trabajo es cada vez menos frecuente por el alto costo de desarrollo y mantenimiento.

2.2 ORM

La mayor parte de las aplicaciones orientadas a objetos precisan de la implementación de mecanismos que permitan a los objetos mantenerse vivos tras la finalización del proceso que les dio vida. El problema de la persistencia de objetos Java ha dado lugar a utilizar alternativas que proporcionan al programador una manera sencilla, automática y transparente basada en bases de datos relacionales, de incluir esa funcionalidad en sus desarrollos denominada, ORM (Mapeo Objeto Relacional).

2.2.1 ¿QUÉ ES ORM?

El mapeo Objeto/Relacional es “la persistencia automatizada y transparente de las tablas en una Base de Datos relacional, usando metadatos que definen el mapeo entre los objetos y la Base de Datos”[5].

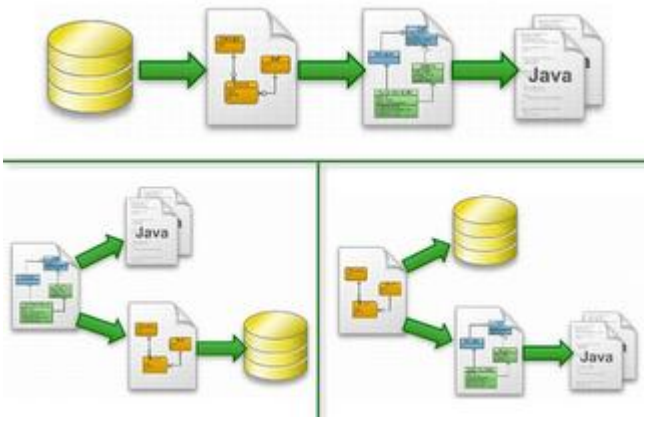


Figura II.6.Conversión de datos

Fuente: http://griseldithaa.blogspot.com/

En esencia, ORM transforma datos de una representación en otra.

Por tanto, ORM es una técnica que se utiliza para poder ligar las bases de datos y los conceptos de orientación a objetos creando ”bases de datos virtuales”, es decir la aplicación desde dentro utiliza Frameworks, los mismos que son intermediarios entre la base de datos relacional y la aplicación totalmente orientada a objetos[6].

2.2.2. COMPONENTES DE UN ORM

Una solución ORM consta de los cuatros aspectos siguientes:

[5]Bauer C; Gavin k. Hibernate in Action Practical Object/Relational Mapping. Greenwich-EE.UU, Maning, ©2004

[6]http://dspace.espoch.edu.ec/bitstream/123456789/1930/1/18T00499.pdf

- Una API para realizar las operaciones básicas CRUD sobre objetos de clases persistentes.

- Un lenguaje o API para especificar consultas que hacen referencia a las clases y sus propiedades.
- Facilidad para especificar mapeo de metadatos.
- Una técnica para que la implementación del ORM pueda llevar a cabo búsquedas, asociaciones y otras funciones de optimización.

Mediante el ORM, la aplicación interactúa con elORM API y las clases del modelo de dominio y se abstrae de la subyacenteSQL / JDBC. Dependiendo de las características o de la implementación particular.

JDBC: Componente indispensable para los ORM

Sin tener en cuenta la solución de mapeo O/R que se vaya a utilizar para comunicarsecon la base de datos relacional, todos ellos dependen de JDBC. Teniendo en cuenta quela mayor parte de las aplicaciones se comunican con bases de datos relacionales, esfundamental considerar cada uno de los niveles del software (desde el código delprograma hasta la fuente de datos) para asegurar que el diseño de persistencia O/R seaóptimo.

Tal y como se verá más adelante, cada una de las soluciones de mapeo O/R tiene unadependencia particular en el driver JDBC para poder comunicarse con la base de datos deuna forma eficiente. Si el driver JDBC que va a participar en la comunicación no esóptimo, la posible gran eficiencia de cualquier Framework quedará debilitada. Por tanto,elegir el driver JDBC que mejor se adapte a la aplicación es esencial a la hora de construirun sistema eficiente en el que participe una solución de mapeo O/R.

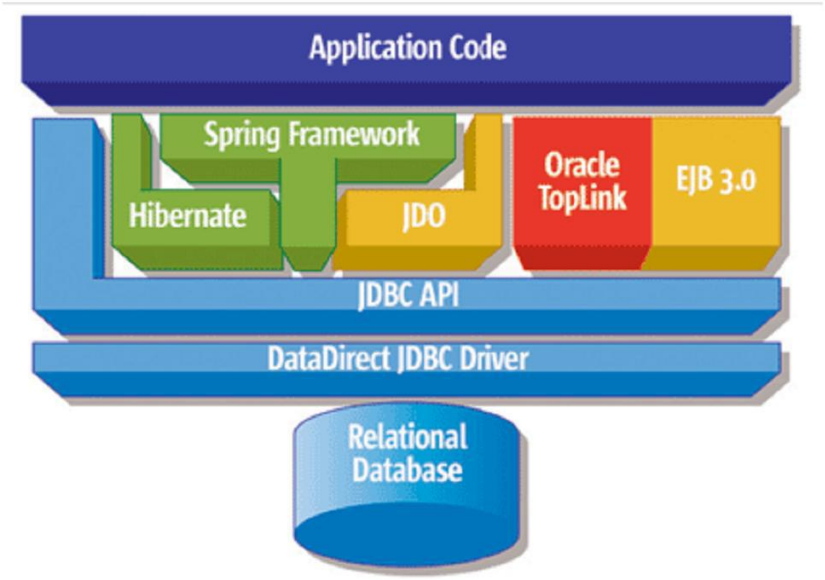


Figura II.7. Interacción de Componentes

Fuente: http://java.sys-con.com/node/140123

La figura muestra una representación de los diferentes mecanismos o soluciones demapeo O/R y cómo se relacionan con el código de la aplicación y con los recursos de datos relacionados. Esto muestra claramente la función crítica que desempeña el driverJDBC puesto que está situado en la base de cada uno de los Frameworks.

La eficiencia del driver JDBC tiene importantes consecuencias en el comportamiento delas aplicaciones. Cada mecanismo de mapeo O/R es completamente dependiente deldriver, sin tener en cuenta el diseño de la API del Framework que esté expuesta al códigofuente de la aplicación.

Como los mecanismos de mapeo O/R generan llamadas eficientes para acceder a la basede datos, mucha gente defiende que la importancia del driver JDBC se ha visto reducida.

Sin embargo, como en cualquier arquitectura, la totalidad de eficiencia en una aplicaciónsiempre estará afectada por el nivel más débil del sistema.

Independientemente del código JDBC generado, los mecanismos de mapeo O/R sonincapaces de controlar cómo los drivers interactúan con la base de datos. Entonces laeficiencia de la aplicación depende en gran parte de la habilidad que tenga el driver delnivel JDBC para mover todos los datos manejados entre la aplicación y la base de datos.

Aunque hay múltiples factores que considerar a la hora de elegir un driver JDBC,seleccionar el mejor driver JDBC posible basándose en comportamiento, escalabilidad yfiabilidad es la clave para obtener el máximo beneficio de cualquier aplicación basada enun Framework O/R.

Mapeando asociaciones

Existen tres tipos de asociaciones entre objetos:

- Asociación
- Agregación
- Composición

Por ahora las trataremos igual, aunque existen diferencias en el manejo de las restricciones. Las cuales se pueden clasificar de acuerdo a dos categorías ortogonales:

En base a la multiplicidad:

- One-to-one
- One-to-many (many-to-one segúndesdedonde se lea).
- Many-to-many

En base a la dirección:

- Unidireccionales
- Bidireccional

En la base de datos las relaciones se mantienen mediante el uso de ForeignKeys:

- One-to-one: FK implementada en una de las tablas.
- One-to-many: FK desde la “one table” a la “many table”
- Many-to-many: Es necesario incorporar una tabla asociativa (o de relación)

En cuanto a la direccionalidad, todas las relaciones en la base de datos relacional sonefectivamente bidireccionales.

Una configuración importante es si los objetos asociados a otro se leen automáticamenteal leer el mismo. Cargar un objeto y todos sus objetos asociados automáticamente enmemoria puede llegar a impactar negativamente en el rendimiento del sistema. Para estose usan técnicas de **lazyloading**que consisten en cargar los objetos asociados ademanda a medida que son solicitados. Si una asociación nunca es solicitada entoncesnunca se lee de la base de dato.

Otra consideración importante tiene que ver con el almacenamiento de coleccionessecuenciales de objetos asociados a un objeto padre. En este caso es necesarioalmacenar en la base la información (por ejemplo un ordinal numérico) que permitarecuperar estos objetos en la secuencia correcta

2.2.3 EL ORM SE PUEDE USAR DE DIFERENTES MANERAS.

MarkFussel [Fussel 1997], un investigador en el campo de ORM, definió los siguientes cuatro niveles decalidad[7].

Relacional puro

La aplicación completa, incluyendo la interfaz de usuario está diseñado entorno al modelo relacional y basadas en operaciones relacionales SQL. Este enfoque, a pesar de sus deficienciaspara sistemas grandes, puede ser una solución excelente para aplicaciones sencillasdonde un bajo nivel de reutilización de código es tolerable. Direct SQL puede ser ajustado en cadaaspecto, pero los inconvenientes, como la falta de

[7] Fussel, Mark L. 1997. Foundations of Object Relational Mapping. ChiMu Corporation.

portabilidad y facilidad de mantenimiento, son significativos, especialmente en el largo plazo. Las aplicaciones de esta categoría suelen ser pesados por el uso de procedimientos almacenados, cambiando algunos de los trabajos de la capa de negocio yen la base de datos.

Mapeo objeto Ligero

Las entidades se representan como clases que son mapeadas manualmente a lastablas relacionales. Codificando a mano SQL / JDBC se oculta de la lógica de negocio utilizando correctamente patrones de diseño conocidos. Este enfoque está muy extendido y tiene éxitopara aplicaciones con un pequeño número de entidades, o con aplicaciones genéricas,metadatos impulsados por modelos de datos. Los procedimientos almacenados pueden tener un lugar en este tipode aplicación.

Mapeo objeto Medio

La aplicación está diseñada alrededor de un modelo de objetos. SQL es generado al momento de construir usando una herramienta de generación de código, o en tiempo de ejecución mediante código Framework. Las asociacionesentre los objetos están soportadas por el mecanismo de persistencia, y se pueden hacer consultasespecíficas utilizando un lenguaje de expresión orientado a objetos. Los objetos se almacenan en caché por la capa de persistencia. Una gran cantidad de productos ORM y las capas de persistenciasoportan al menos este nivel de funcionalidad. Es muy adecuado para aplicaciones de tamaño mediocon algunas operaciones complejas, sobre todo cuando la portabilidad entre diferentes productos de base de datos es importante. Estas aplicaciones normalmente no utilizanprocedimientos almacenados.

Mapeo objeto Completo

Mapeo objeto Full soporta modelado de objetos sofisticados: composición, herencia, polimorfismo, y la "persistencia por alcance." La capa de persistencia implementa persistencia transparente; clases persistentes no heredan ninguna clase

base especial o tener que implementar una interfaz especial. Estrategias eficientes de búsqueday de almacenamiento en caché las cuales se llevan a cabo de forma transparente parala aplicación. Este nivel de funcionalidad difícilmente se puede lograr mediante una capa de persistencia homegrown. Un gran número de herramientas ORM de Java han alcanzado este nivel decalidad.

2.2.4. VENTAJAS DE UTILIZAR UN ORM

Rapidez en el desarrollo: La mayoría de las herramientas actuales permiten la creación del modelo por medio del esquema de la base de datos, leyendo el esquema, nos crea el modelo adecuado.

Abstracción de la base de datos: Al utilizar un sistema ORM, lo que conseguidos es separarnos totalmente del sistema de Base de datos que utilicemos, y así si en un futuro debemos de cambiar de motor de bases de datos, tendremos la seguridad de que este cambio no nos afectará a nuestro sistema, siendo el cambio más sencillo.

Reutilización: Nos permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, incluso desde aplicaciones distintas.

Seguridad: Los ORM suelen implementar sistemas para evitar tipos de ataques como los SQL injections.

Mantenimiento del código: Nos facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.

Lenguaje propio para realizar las consultas: Estos sistemas de mapeo traen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar las sentencias SQL para que pasen a utilizar el lenguaje propio de cada herramienta.

2.2.5. DESVENTAJAS DE UTILIZAR UN ORM

Tiempo utilizado en el aprendizaje: Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.

Aplicaciones algo más lentas: Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

En la actualidad hay muchos tipos de Framework que nos devuelven el mapeo objeto-relacional, según el lenguaje que estemos utilizando. Vamos a nombrar algunos de los más utilizados.

- Doctrine
- Propel
- Hibernate
- Linq

CAPÍTULO III

3. FRAMEWORKHIBERNATE

3.1 INTRODUCCIÓN.

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Hibernate es un Framework que agiliza la relación entre la aplicación y la base de datos. De todos los Frameworks ORM que se ha utilizado, sin dudas es el más completo.

Para aprender Hibernate es necesario tener los conocimientos mínimos de SQL y Java. Conocer JDBC es recomendable.

3.2 HISTORIA DE HIBERNATE

Hibernate fue una iniciativa de un grupo de desarrolladores dispersos alrededor del mundo conducidos por Gavin King.Tiempo después, JBoss Inc. (empresa comprada por Red Hat) contrató a los principales desarrolladores de Hibernate y trabajó con ellos en brindar soporte al proyecto.

La versión actual de desarrollo de Hibernate es la 4.x, la cual incorpora nuevas características, como una nueva arquitectura Interceptor/Callback, filtros definidos por el usuario, y opcionalmente el uso de anotaciones para definir la correspondencia en lugar de (o conjuntamente con) los archivos XML.

3.3 QUE ES HIBERNATE

Hibernate es una capa de persistencia objeto/relacional y un generador de sentencias SQL que permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada se puede generar BBDD en cualquiera de los entornos soportados: Oracle, DB2, MySql, PostgreSql, etc. Y lo mas importante de todos, es open source, lo que supone, entre otras cosas que no se tiene que pagar ningún costo para adquirirlo.

Uno de los posibles procesos de desarrollo consiste en, que una vez que se tenga el diseño de datos realizado, se mapea este a ficheros XML siguiendo la DTD de mapeo de Hibernate.

Desde estos se puede generar el código de nuestros objetos persistentes en clases Java y también crear BBDD independientemente del entorno escogido.

Hibernate se integra en cualquier tipo de aplicación justo por encima del contenedor de datos. Una posible configuración básica de Hibernate es la siguiente:

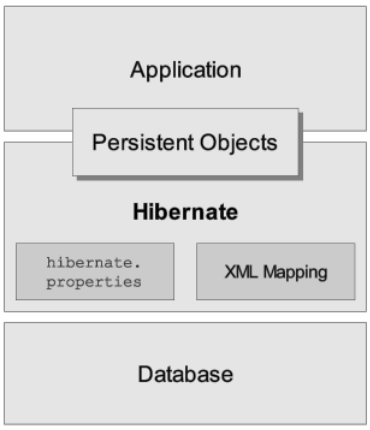


Figura III.8. Arquitectura Base de Hibernate

Fuente:http://docs.jboss.org/hibernate/orm/3.5/reference/es-es/html/architecture.html

Se puede observar en la figura.III.8como Hibernate utiliza la BBDD y la configuración de los datos para proporcionar servicios y objetos persistentes a la aplicación que se encuentre justo por arriba de él.

3.4 ¿POR QUÉ EL USO DE ESTE FRAMEWORK?

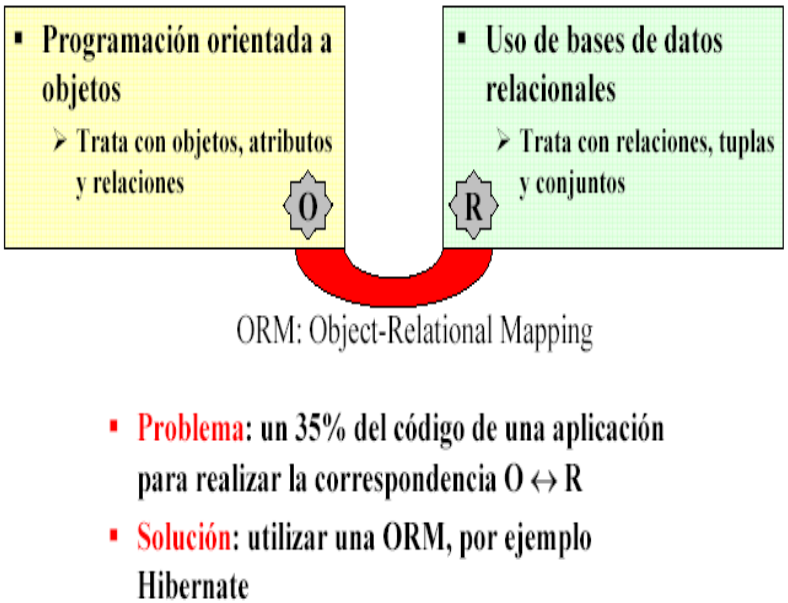


FIGURA III.9.USO DEL FRAMEWORKHIBERNATE

Fuente: http%3A%2F%2Fwww.unife.edu.pe%2Fing%2Fdesarrollo.doc&ei

Si se está trabajando con programación orientada a objetos y bases de datos relacionales, se observa que son dos paradigmas diferentes.

El modelo relacional trata con relaciones, tuplas y conjuntos y es muy matemático por naturaleza. Sin embargo, el paradigma orientado a objetos trata con objetos, sus atributos y relaciones entre objetos. Cuando se quiere hacer que los objetos sean persistentes utilizando para ello una base de datos relacional, uno se da cuenta de que hay una desavenencia entre estos dos paradigmas, la también llamada diferencia objeto-relacional (object – relational gap"). Un mapeador objeto-relacional **(ORM)** nos ayudará a evitar esta diferencia.

¿Cómo se manifiesta esta brecha entre ambos paradigmas?

Si estamos utilizando objetos en nuestra aplicación y en algún momento queremos que sean persistentes, normalmente abriremos una conexión JDBC, crearemos una sentencia SQL y copiaremos todos los valores de las propiedades sobre una PreparedStatement o en la cadena SQL que estemos construyendo. Esto podría ser fácil para un objeto de tipo valor (value object:VO) de pequeño tamaño pero consideremos esto para un objeto con muchas propiedades. Este no es el único problema.

¿Qué pasa con las asociaciones? ¿Y si el objeto contiene a su vez a otros objetos? ¿Los almacenaremos también en la Base de Datos? ¿Automáticamente? ¿Manualmente? ¿Qué haremos con las claves ajenas? Preguntas similares surgen a la hora de "cargar" un dato de la BD de un VO (se denomina **value object** o **VO** a un objeto que contiene información de negocio estructurada en grupos de ítems de datos, también recibe el nombre de transfer object).

Como se puede ver, la brecha existente entre los paradigmas de objeto y relacional se vuelve mucho mayor si disponemos de modelos con objetos "grandes". Y hay muchas más cosas a considerar como la carga lenta, las referencias circulares, el caché, etc.

De hecho, hay estudios que demuestran que el 35% del código de una aplicación se produce como consecuencia del mapeado (correspondencia) entre los datos de la aplicación y el almacén de datos.

Entonces, lo que necesitamos es una herramienta **ORM** (Object Relational Mapping). Básicamente, una ORM intenta hacer todas estas tareas pesadas por nosotros. Con una buena ORM, sólo tendremos que definir la forma en la que establecemos la correspondencia entre las clases y las tablas una sola vez (indicando que propiedad se corresponde con que columna, que clase con que tabla, etc.). Después de esto, podremos hacer cosas como utilizar **POJO'**s (Plain Old Java Objects) de nuestra aplicación y decirle a nuestra ORM que los haga persistentes, con una instrucción similar a esta: orm.save (myObject). Es decir, una herramienta puede leer o escribir en la base de datos utilizando VO's directamente.

Más formalmente: un modelo del dominio representa las entidades del negocio utilizadas en una aplicación Java. En una arquitectura de sistemas por capas, el modelo del dominio se utiliza para ejecutar la logica del negocio (en Java, no en la base de datos). Esta capa del negocio se comunica con la capa de persistencia subyacente para recuperar y almacenar los objetos persistentes del modelo del dominio. ORM es el middleware en la capa de persistencia que gestiona la persistencia.

Hibernate es un ORM de libre distribución, que además, es de las más maduras y completas. Actualmente su uso esta muy extendido y además esta siendo desarrollada de forma muy activa. Una característica muy importante que distingue Hibernate de otras soluciones al problema de la persistencia, como los Ejes de entidad, es que la clase Hibernate persistente puede utilizarse en cualquier contexto de ejecución, es decir, no se necesita un contenedor especial para ello.

3.5 OBJETIVOS DE HIBERNATE

El objetivo de Hibernate es liberar al desarrollador del 95% de las tareas deprogramación comunes relacionadas con la persistencia de datos en comparacióncon una codificación manual con SQLy el API JDBC.

Como todas las herramientas de su tipo, Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución. Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible. Hibernate ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria"). Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente Hibernate Annotations que implementa el estándar JPA, que es parte de esta plataforma.

3.6 CARACTERÍSTICAS DE HIBERNATE

- No intrusivo (estilo POJO)
- Muy buena documentación (forums para ayuda, libro)
- Comunidad activa con muchos usuarios
- Transacciones, caché, asociaciones, polimorfismo, herencia, lazy loading, persistencia transitiva, estrategias de fetching.
- Potente lenguaje de consulta (HQL): subqueries, outer joins, ordering, proyeccion (report query), paginacion.
- Facil testeo.
- No es standard.

3.7 ARQUITECTURA

El API de Hibernate es una arquitectura de dos capas (Capa de persistencia y Capa de Negocio).

El diagrama a continuación brinda una perspectiva a alto nivel de la arquitectura de Hibernate:

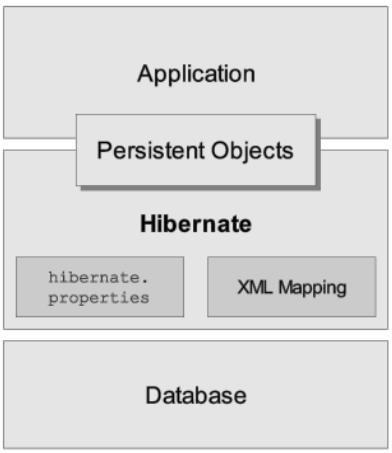


Figura III.10.Arquitectura de alto nivel deHibernate

Fuente:http://docs.jboss.org/Hibernate/orm/3.5/reference/es-ES/html/architecture.html

Hibernate es flexible y soporta diferentes enfoques. Sin embargo, mostraremos los dos extremos: la arquitectura "mínima" y la arquitectura "completa".

Este diagrama ilustra la manera en que Hibernate utiliza la base de datos y los datos de configuración para proporcionar servicios de persistencia y objetos persistentes a la aplicación.

La arquitectura "mínima" hace que la aplicación proporcione sus propias conexiones JDBC y que administre sus propias transacciones. Este enfoque utiliza un subgrupo mínimo de las APIs de Hibernate:

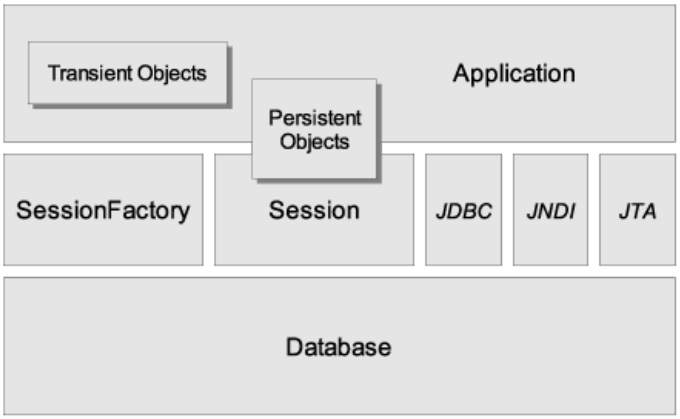


Figura III.11.Arquitectura mínima de Hibernate

Fuente:http://docs.jboss.org/Hibernate/orm/3.5/reference/es-ES/html/architecture.html

La arquitectura "completa" abstrae la aplicación de las APIs de JDBC/JTA y permite que Hibernate se encargue de los detalles.

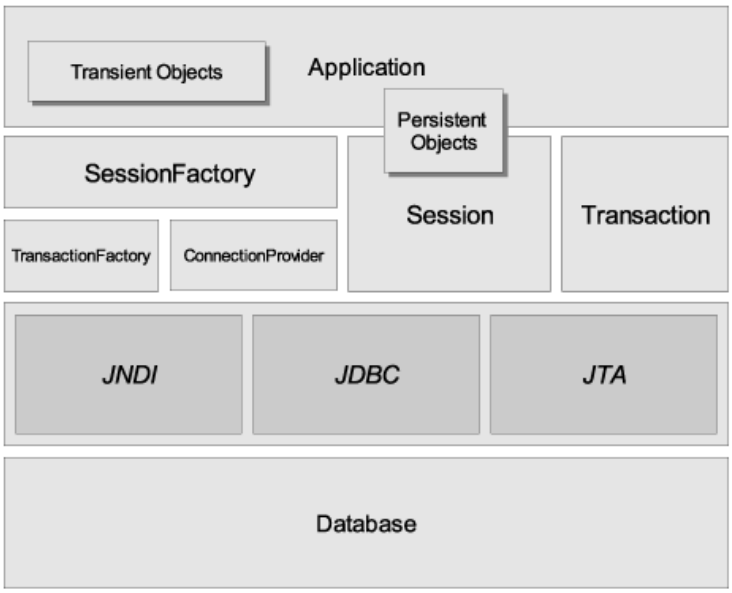


Figura III.12. Arquitectura completa de Hibernate

Fuente: http://docs.jboss.org/Hibernate/orm/3.5/reference/es-ES/html/architecture.html

He aquí algunas definiciones de los objetos en los diagramas:

- **SessionFactory (org.Hibernate.SessionFactory)**

Un caché threadsafe (inmutable) de mapeos compilados para una sola base de datos. Una fábrica de Session y un cliente de ConnectionProvider, SessionFactory puede mantener un caché opcional (de segundo nivel) de datos reusables entre transacciones a nivel de proceso o de clúster.

- **Session (org.Hibernate.Session)**

Un objeto mono-hebra, de corta vida que representa una conversación entre la aplicación y el almacenamiento persistente. Envuelve una conexión JDBC y es una fábrica de Transaction. Session mantiene un caché requerido de primer nivel de objetos persistentes, que se utiliza cuando se navega el gráfico de objetos o mientras se buscan objetos por identificador.

Objetos y colecciones persistentes

Objetos de corta vida, mono-hebra contienen un estado persistente así como una funcionalidad empresarial. Estos pueden ser JavaBeans/POJOs normales. Estos se encuentran asociados con exactamente una Session. Tan pronto como la Session se cierre, serán separados y estarán libres para utilizarlos en cualquier capa de aplicación, (por ejemplo, directamente como objetos de transferencia de datos hacia y desde la presentación).

Objetos y colecciones transitorias y separadas

Instancias de clases persistentes que no se encuentran actualmente asociadas con una Session. Pueden haber sido instanciadas por la aplicación y aún no haber sido persistidas, o pueden haber sido instanciadas por una Session cerrada.

- **Transaction (org.Hibernate.Transaction)**

(Opcional) Un objeto de corta vida, mono-hebra que la aplicación utiliza para especificar unidades atómicas de trabajo. Abstrae la aplicación de las transacciones subyacentes JDBC, JTA o CORBA. En algunos casos, una Session puede extenderse sobre varias Transactiones. Sin embargo, la demarcación de la transacción, ya sea utilizando la API subyacente o Transaction, nunca es opcional.

- **ConnectionProvider (org.Hibernate.connection.ConnectionProvider)**

(Opcional) Una fábrica y pool de conexiones JDBC. Abstrae a la aplicación del Datasource o DriverManager subyacente. No se expone a la aplicación, pero puede ser extendido/implementado por el desarrollador.

- **TransactionFactory (org.Hibernate.TransactionFactory)**

(Opcional) Una fábrica de instancias de Transaction. No se expone a la aplicación pero puede ser extendido/implementado por el desarrollador.

Extensión Interfaces

Hibernate ofrece un rango de interfaces de extensión opcionales que puede implementar para personalizar el comportamiento de su capa de persistencia. Para obtener más detalles, vea la documentación de la API.

Dada una arquitectura "sencilla", la aplicación evita las APIs de Transaction/TransactionFactory y/o ConnectionProvider, para comunicarse directamente con JTA o JDBC.

3.7.1 ESTADOS DE INSTANCIA

Una instancia de una clase persistente puede estar en uno de tres estados diferentes. Estos estados se definen con respecto a su *contexto de persistencia*. El objeto Session de Hibernate es el contexto de persistencia. Los tres estados diferentes son los siguientes:

- **Transitorio**

La instancia no está asociada con un contexto de persistencia. No tiene identidad persistente o valor de clave principal.

- **Persistente**

La instancia se encuentra actualmente asociada con un contexto de persistencia. Tiene una identidad persistente (valor de clave principal) y puede tener una fila correspondiente en la base de datos. Para un contexto de persistencia en particular,

Hibernate*garantiza* que la identidad persistente es equivalente a la identidad Java en relación con la ubicación del objeto.

- **Separado**

La instancia estuvo alguna vez asociada con un contexto de persistencia, pero ese contexto se cerró, o la instancia fue serializada a otro proceso. Tiene una identidad persistente y puede tener una fila correspondiente en la base de datos. Para las instancias separadas, Hibernate no establece ninguna garantía sobre la relación entre identidad persistente e identidad Java.

3.7.2 INTEGRACIÓN JMX

JMX es el estándar J2EE para la gestión de componentes Java. Hibernate se puede administrar por medio de un servicio estándar JMX. Brindamos una implementación de MBean en la distribución: org.Hibernate.jmx.HibernateService.

- **Administración de Sesión:** El ciclo de vida de la Session de Hibernate puede estar ligado automáticamente al ámbito de una transacción JTA. Esto significa que ya no tiene que abrir ni cerrar la Session manualmente, esto pasa a ser el trabajo de un interceptor EJB de JBoss. Además tampoco tiene que preocuparse más de la demarcación de la transacción en su código (a menos de que quiera escribir una capa de persitencia portátil, utilice la API de Transaction de Hibernate para hacer esto). Para acceder a una Session llame al HibernateContext.
- **Despliegue HAR:** el servicio JMX de Hibernate se implementa usando un descriptor de despliegue de servicio de JBoss en un archivo EAR y/o SAR, que soporta todas las opciones de configuración usuales de una SessionFactory de Hibernate. Sin embargo, todavía tiene que nombrar todos sus archivos de

mapeo en el descriptor de despliegue. Si utiliza el depliegue HAR opcional, JBoss detectará automáticamente todos los archivos de mapeo en su archivo HAR.

3.7.3 SOPORTE JCA

Hibernate también puede ser configurado como un conector JCA. Sin embargo, tenga en cuenta que el soporte de JCA de Hibernate aún está bajo desarrollo.

3.7.4 SESIONES CONTEXTUALES

La mayoría de las aplicaciones que utilizan Hibernate necesitan alguna forma de sesiones "contextuales", en donde una sesión dada se encuentra en efecto en todo el campo de acción de un contexto dado. Sin embargo, a través de las aplicaciones la definición de lo que constituye un contexto es usualmente diferente y diferentes contextos definen diferentes campos de acción para la noción de actual. Las aplicaciones que utiliza Hibernate antes de la version 3.0 tienden a utilizar ya sea sesiones contextuales con base ThreadLocal desarrollados en casa, las clases ayudantes tales como HibernateUtil, o enfoques de terceros utilizados, como Spring o Pico, los cuales brindaban sesiones contextuales con base proxy/intercepción.

Comenzando con la version 3.0.1, Hibernate agregó el método SessionFactory.getCurrentSession(). Inicialmente, este asumió la utilización de las transacciones JTA, en donde la transacción JTA definia tanto el contexto como el campo de acción de una sesión actual. Dada la madurez de números osas implementaciones JTA TransactionManager autónomas existentes, la mayoría, si no es que todas, las aplicaciones deberían utilizar la administración de transacciones JTA en el caso de que se deplieguen o no en un contenedor J2EE. Con base en esto, las sesiones contextuales basadas en JTA es todo lo que usted necesita utilizar.

Sin embargo, desde la versión 3.1, el procesamiento detrás de SessionFactory.getCurrentSession() ahora es conectable. Para ese fin, se ha añadido una nueva interfaz de extensión, org.Hibernate.context.CurrentSessionContext, y un nuevo parámetro de configuración, Hibernate.current_session_context_class para permitir la conexión del campo de acción y el contexto de definición de las sesiones actuales.

Refiérase a los Javadocs para la interfaz org.Hibernate.context.CurrentSessionContext para poder ver una discusión detallada de su contrato. Define un método único, currentSession(), por medio del cual la implementación es responsable de rastrear la sesión contextual actual. Tal como viene empacada, Hibernate incluye tres implementaciones de esta interfaz:

- **org.Hibernate.context.JTASessionContext:** una transacción JTA rastrea y asume las sesiones actuales. Aquí el procesamiento es exactamente el mismo que en el enfoque más antiguo de JTA-solamente. Refiérase a los Javadocs para obtener más información.
- **org.Hibernate.context.ThreadLocalSessionContext:** las sesiones actuales son rastreadas por un hilo de ejecución. Consulte los Javadocs para obtener más detalles.
- **org.Hibernate.context.ManagedSessionContext:** las sesiones actuales son rastreadas por un hilo de ejecución. Sin embargo, usted es responsable de vincular y desvincular una instancia Session con métodos estáticos en esta clase: no abre, vacia o cierra una Session.

El parámetro de configuración Hibernate.current_session_context_class define cuales implementaciones org.Hibernate.context.CurrentSessionContext deben utilizarse. Para compatibilidad con versiones anteriores, si este parámetro de configuración no está establecido pero si tiene configurado unorg.Hibernate.transaction.TransactionManagerLookup, Hibernate utilizará el

org.Hibernate.context.JTASessionContext. Usualmente el valor de este parámetro sólamente nombraría la clase de implementación a utilizar. Sin embargo, para las tres implementaciones incluidas existen tres nombres cortos: "jta", "thread" y "managed".

3.8 Ciclo de Vida de la persistencia

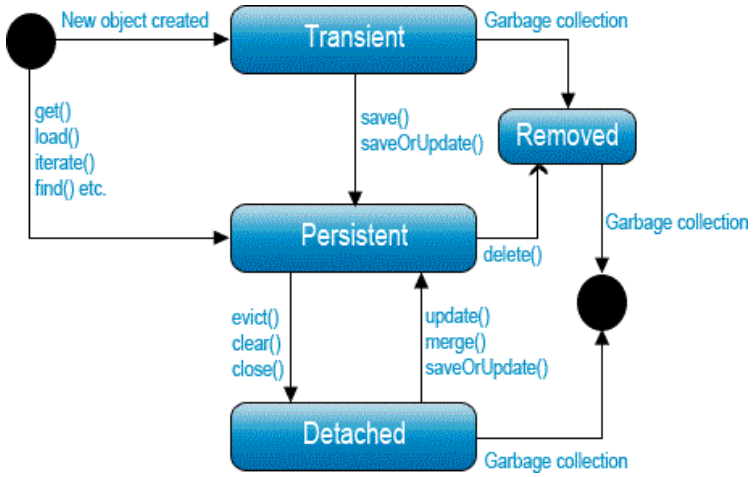


Figura III.13.Ciclo de vida de persistencia de Hibernate.

Fuente: http://www.roseindia.net/hibernate/hibernatelifecycle.shtml

- **Estados**
  - Transient (Transitorio)
  - Objetos recién creados
  - Persistent (Persistente)
  - Objetos salvados (save o persist)
  - Objetos cargados (get, find, load, list, etc)
  - Objetos ‘re-enlazados’ (update, merge)
  - Detached (Desenlazados)
  - Después de cerrar el Session/EntityManager
  - Removed (Removidos)
  - Objetos borrados (delete/remove)
- **Persistence Context**

- Conjunto de entidades administradas
- Asegura unicidad de entidades persistentes
- Detecta y administra las modificaciones realizadas a entidades persistentes
- Sirve como una cache de primer nivel
- Automatic dirty checking

➢ **Detección y administración de cambios a las entidadespersistentes manejadas**

- Hibernate posterga la actualización de la base de datoshasta el último momento posible dentro de latransacción
- Se utilizan todos los campos en el SQL, salvo que seespecifique dynamic-update / dynamic-insert en elmapeo de la clase

➢ **Identidad**

- Identidad de persistencia
- Primary Key
- @Id
- Session.getId()
- Identidad de objetos
- equals()
- hashCode()
- Identidad de dominio (business identity) campos que identifican cada entidad en el dominioconsiderado (puede incluir el id o no)

➢ **Conversaciones**

- Unidad de trabajo (Unit of work) extendida
- Permite reenlazar objetos para volverlos persistentes otra vez
- Dos estrategiassesión por pedido con objetos desenlazados (session per request with detachedobjects)

- **Contexto de persistencia extendido (session per conversation)**

3.8.1 Clases persistentes

Las clases persistentes son clases en una aplicación que implementan las entidades del problema empresarial (por ejemplo, Customer y Order en una aplicación de comercio electrónico). No se considera que todas las instancias de una clase persistente estén en estado persistente. Por ejemplo, una instancia puede ser transitoria o separada.

Hibernate funciona mejor si estas clases siguen algunas reglas simples, también conocidas como el modelo de programación POJO (Plain Old Java Object). Sin embargo, ninguna de estas reglas son requerimientos rígidos. De hecho, Hibernate3 asume muy poco acerca de la naturaleza de sus objetos persistentes. Puede expresar un modelo de dominio en otras formas (por ejemplo, utilizando árboles de instancias de Map).

3.8.2 EJEMPLO SIMPLE DE POJO

La mayoría de aplicaciones Java requieren una clase persistente que represente a los felinos. Por ejemplo:

```
package eg;
import java.util.Set;
import java.util.Date;

public class Cat {
    private Long id; // identifier

    private Date birthdate;
    private Color color;
    private char sex;
    private float weight;
    private int litterId;
```

```
    private Cat mother;
    private Set kittens = new HashSet();

    private void setId(Long id) {
        this.id=id;
    }
    public Long getId() {
        return id;
    }

    void setBirthdate(Date date) {
        birthdate = date;
    }
    public Date getBirthdate() {
        return birthdate;
    }

    void setWeight(float weight) {
        this.weight = weight;
    }
    public float getWeight() {
        return weight;
    }

    // addKitten not needed by Hibernate
    public void addKitten(Cat kitten) {
            kitten.setMother(this);
        kitten.setLitterId( kittens.size() );
        kittens.add(kitten);
    }
}
```

3.8.3 IMPLEMENTACION UN CONSTRUCTOR SIN ARGUMENTOS

Cat tiene un contructor sin argumentos. Todas las clases persistentes deben tener un constructor predeterminado (el cual puede ser no-público) de modo que Hibernate pueda instanciarlas usando Constructor.newInstance(). Le recomendamos contar con un constructor por defecto con al menos una visibilidad de *paquete* para la generación de proxies en tiempo de ejecución en Hibernate.

3.8.4 DECLARACION DE MÉTODOS DE ACCESO Y DE MODIFICACIÓN PARA LOS CAMPOS PERSISTENTES (OPCIONAL)

Cat declara métodos de acceso para todos sus campos persistentes. Muchas otras herramientas ORM persisten directamente variables de instancia. Es mejor proporcionar una indirección entre el esquema relacional y las estructuras de datos internos de la clase. Por defecto, Hibernate persiste las propiedades del estilo JavaBeans, y reconoce los nombres de método de la forma getFoo, isFoo y setFoo. De ser necesario, puede cambiarse al acceso directo a campos para propiedades específicas.

No es necesario declarar públicas las propiedades. Hibernate puede persistir una propiedad con un par get / set protected o private.

3.8.5 IMPLEMENTACIÓN DE HERENCIA

Una subclase también tiene que cumplir con la primera y la segunda regla. Hereda su propiedad identificadora de la superclase Cat. Por ejemplo:

```
package eg;
public class DomesticCat extends Cat {
        private String name;
        public String getName() {
                return name;
        }
        protected void setName(String name) {
                this.name=name;
        }
}
```

3.8.6 MODELOS DINÁMICOS

Las entidades persistentes no necesariamente tienen que estar representadas como clases POJO o como objetos JavaBean en tiempo de ejecución. Hibernate también

soporta modelos dinámicos (utilizando Mapeos de Mapeos en tiempo de ejecución) y la representación de entidades como árboles de DOM4J. No escriba clases persistentes con este enfoque, sólamente archivos de mapeo.

Los siguientes ejemplos demuestran la representación utilizando Mapeos. Primero, en el archivo de mapeo tiene que declararse un entity-name en lugar de, o además de un nombre de clase:

```
<Hibernate-mapping>
<class entity-name="Customer">

    <id name="id"
        type="long"
        column="ID">
        <generator class="sequence"/>
    </id>
    <property name="name"
        column="NAME"
        type="string"/>
    <property name="address"
        column="ADDRESS"
        type="string"/>
    <many-to-one name="organization"
        column="ORGANIZATION_ID"
        class="Organization"/>
    <bag name="orders"
        inverse="true"
        lazy="false"
        cascade="all">
        <key column="CUSTOMER_ID"/>
        <one-to-many class="Order"/>
    </bag>
</class>
</Hibernate-mapping>
```

Aunque las asociaciones se declaran utilizando nombres de clase destino, el tipo destino de una asociación puede ser además una entidad dinámica en lugar de un POJO.

Después de establecer el modo de entidad predeterminado como dynamic-map para la SessionFactory, puede trabajar en tiempo de ejecución con Mapeos de Mapeos:

```
Session s = openSession();
Transaction tx = s.beginTransaction();
// Create a customer
Map david = new HashMap();
david.put("name", "David");
// Create an organization
Map foobar = new HashMap();
foobar.put("name", "Foobar Inc.");
// Link both
david.put("organization", foobar);
// Save both
s.save("Customer", david);
s.save("Organization", foobar);
tx.commit();
s.close();
```

Una de las ventajas principales de un mapeo dinámico es el rápido tiempo de entrega del prototipado sin la necesidad de implementar clases de entidad. Sin embargo, pierde el chequeo de tipos en tiempo de compilación y muy probablemente tendrá que tratar con muchas excepciones en tiempo de ejecución. Gracias al mapeo de Hibernate, el esquema de base de datos se puede normalizar y volver sólido, permitiendo añadir una implementación apropiada del modelo de dominio más adelante.

Los modos de representación de entidad se pueden establecer por Session:

```
Session dynamicSession = pojoSession.getSession(EntityMode.
MAP);

// Create a customer
Map david = new HashMap();
david.put("name", "David");
dynamicSession.save("Customer", david);
...
dynamicSession.flush();
dynamicSession.close()
...
// Continue on pojoSession
```

Tenga en cuenta que la llamada a getSession() utilizando un EntityMode está en la API de Session, no en la de SessionFactory. De esta forma, la nueva Session

comparte la conexión JDBC, la transacción y otra información de contexto. Esto significa que no tiene que llamar a flush() ni a close() en la Session secundaria, y también tiene que dejar el manejo de la transacción y de la conexión a la unidad de trabajo primaria.

3.9 Herramientas de Hibernate

Las herramientas de Hibernate actualmente incluyen plugins la IDE de Eclipse así como tareas Ant para la ingeniería inversa de bases de datos existentes:

- **Editor de Mapeo:** Un editor de archivos de mapeo XML que soporta autocompleción y resaltado de sintáxis. También soporta la autocompleción semántica de nombres de clases y nombres de campos/propiedades, haciéndolo mucho más versátil que un editor normal de XML.
- **Consola:** La consola es una nueva vista en Eclipse. Además de la vista de árbol de sus configuraciones de la consola, también tiene una vista interactiva de sus clases persistentes y sus relaciones. La consola le permite ejecutar consultas HQL en su base de datos y navegar el resultado directamente en Eclipse.
- **Asistentes de desarrollo:** Se proporcionan muchos asistentes junto con las herramientas Eclipse de Hibernate. Puede utilizar un asistente para generar rápidamente los archivos de configuración de Hibernate (cfg.xml), o incluso puede realizar una ingeniería inversa completa de un esquema de la base de datos existente en los archivos de código fuente de POJO y los archivos de mapeo de Hibernate. El asistente de ingeniería inversa soporta plantillas personalizables.

Sin embargo, el paquete principal de Hibernate viene con una herramienta integrada: SchemaExport también conocida como hbm2ddl. Incluso se puede utilizar "dentro" de Hibernate.

3.9.1 GENERACIÓN AUTOMÁTICA DE ESQUEMAS

Una de las funciones de Hibernate puede generar DDL desde sus archivos de mapeo. El esquema generado incluye restricciones de integridad referencial, claves principales y foráneas, para las tablas de entidades y colecciones. También se creen tablas y secuencias para los generadores de identificadores mapeados.

Tiene que especificar un Dialecto SQL por medio de la propiedad Hibernate.dialect al usar esta herramienta, ya que el DDL es altamente específico de acuerdo con el vendedor.

Primero, debe personazar sus archivos de mapeo para mejorar el esquema generado. La siguiente sección aborda la personalización del esquema.

- **Personalización del esquema**

Muchos elementos de mapeo de Hibernate definen atributos opcionales denominados length, precision y scale. Con estos atributos puede establecer el tamaño, la precisión y la escala de una columna.

```
<property name="zip" length="5"/>

<property name="balance" precision="12" scale="2"/>
```

Algunas etiquetas también aceptan un atributo not-null para generar una restricción NOT NULL en columnas de tablas y un atributo unique para generar restricciones UNIQUE en columnas de tablas.

```
<many-to-one name="bar" column="barId" not-null="true"/>

<element column="serialNumber" type="long" not-null="true" unique="true"/>
```

Se puede usar un atributo unique-key para agrupar columnas en una restricción de clave única. Actualmente, el valor especificado del atributo unique-key*no* se utiliza

para nombrar la restricción en el DDL generado. Sólamente se utiliza para agrupar las columnas en el archivo de mapeo.

```
<many-to-one name="org" column="orgId" unique-key="OrgEmployeeId"/>
<property name="employeeId" unique-key="OrgEmployee"/>
```

Un atributo index especifica el nombre de un índice que se creará utilizando la columa o las columnas mapeadas. Se pueden agrupar múltiples columnas bajo el mismo índice, simplemente especificando el mismo nombre de índice.

```
<property name="lastName" index="CustName"/>
<property name="firstName" index="CustName"/>
```

Un atributo foreign-key se puede utilizar para sobreescribir el nombre de cualquier restricción de clave foránea generada.

```
<many-to-one name="bar" column="barId" foreign-key="FKFooBar"/>
```

Muchos elementos de mapeo también aceptan un elemento <column> hijo. Esto es particularmente útil para mapear tipos de multi-columna:

```
<property name="name" type="my.customtypes.Name"/>
   <column name="last" not-null="true" index="bar_idx" length="30"/>
   <column name="first" not-null="true" index="bar_idx" length="20"/>
   <column name="initial"/>
</property>
```

El atributo default le permite especificar un valor por defecto para una columna. Usted le debe asignar el mismo valor a la propiedad mapeada antes de guardar una nueva instancia de la clase mapeada.

```
<property name="credits" type="integer" insert="false">
   <column name="credits" default="10"/>
</property>

<version name="version" type="integer" insert="false">
   <column name="version" default="0"/>
</property>
```

El atributo sql-type permite al usuario sobrescribir el mapeo por defecto de tipo Hibernate a tipo de datos SQL.

```
<property name="balance" type="float">
   <column name="balance" sql-type="decimal(13,3)"/>
</property>
```

El atributo check le permite especificar una comprobación de restricción.

```
<property name="foo" type="integer">
   <column name="foo" check="foo> 10"/>
</property>
<class name="Foo" table="foos" check="bar < 100.0">
   ...
   <property name="bar" type="float"/>
</class>
```

La siguiente tabla resume estos atributos opcionales.

Tabla III.I. Resumen de atributos

Atributo	**Valores**	**Interpretación**
length	Número	longitud de columna/precisión decimal
precision	Número	precisión decimal de columna
scale	Número	escala decimal de columna
not-null	true|false	especifica que la columna debe ser sin nulos
unique	true|false	especifica que la columna debe tener una restricción de

Atributo	Valores	Interpretación
		unicidad
index	index_name	especifica el nombre de un índice (multicolumna)
unique-key	unique_key_name	especifica el nombre de una restricción de unicidad multicolumna
foreign-key	foreign_key_name	Especifica el nombre de la restricción de clave foránea generada por una asociación, para un elemento de mapeo <one-to-one>, <many-to-one>, <key>, o <many-to-many>. Observe que SchemaExport no considerará los lados inverse="true".
sql-type	SQL column type	sobrescribe el tipo de columna por defecto (sólamente el atributo del elemento <column>)
default	expresión SQL	especifica un valor predeterminado para la columna
check	expresión SQL	crea una restricción de comprobación SQL en columna o tabla

Fuente: http://docs.jboss.org/Hibernate/core/3.6/reference/es-ES/html_single/

El elemento <comment> le permite especificar un comentario para el esquema generado.

```
<class name="Customer" table="CurCust">
   <comment
>Current customers only</comment>
  </class>
<property name="balance">
   <column name="bal">
      <comment
>Balance in USD</comment>
   </column>
</property>
```

Esto da como resultado una declaración comment on table o comment on column en el DDL generado, donde se encuentre soportado.

- **Ejecución de la herramienta**

La herramienta SchemaExport escribe un script DDL a la salida estándar y/o ejecuta las declaraciones DDL.

La siguiente tabla presenta las opciones de la línea de comandos de SchemaExportJava -cp *Hibernate_classpaths* org.Hibernate.tool.hbm2ddl.SchemaExport *options mapping_files*

Tabla III.II. Opciones de Linea de Comandos de SchemaExport

Opción	Descripción
--quiet	no envie el script a la salida estándar
--drop	sólamente desechar las tablas
--create	sólamente crear las tablas
--text	no exportar a la base de datos
--output=my_schema.ddl	enviar la salida del script ddl a un archivo
--naming=eg.MyNamingStrategy	seleccione un NamingStrategy
--config=Hibernate.cfg.xml	lee la configuración de Hibernate de un archivo XML
--properties=Hibernate.properties	lee las propiedades de base de datos de un archivo
--format	formatea muy bien el SQL generado en el script
--delimiter=;	establece un delimitador de fin de línea para el script

Fuente:http://docs.jboss.org/Hibernate/core/3.6/reference/es-ES/html_single/

- **Propiedades**

Las propiedades de la base de datos se pueden especificar:

- como propiedades del sistema con -D<*property*>
- en Hibernate.properties
- en un archivo de propiedades nombrado con --properties

Las propiedades necesarias son las siguientes:

Tabla III.III. Propiedades de Conexión del Shema Export

Nombre de la Propiedad	**Descripción**
Hibernate.connection.driver_class	clase del controlador jdbc
Hibernate.connection.url	url de jdbc
Hibernate.connection.username	usuario de la base de datos
Hibernate.connection.password	contraseña del usuario
Hibernate.dialect	Dialecto

Fuente: http://docs.jboss.org/Hibernate/core/3.6/reference/es-ES/html_single/

3.10 MAPEO O/R BÁSICO

3.10.1 DECLARACIÓN DE MAPEO

Los mapeos objeto/relacional usualmente se definen en un documento XML. El documento de mapeo está diseñado para que se pueda leer y editar a mano. El lenguaje de mapeo está centrado en Java, lo que significa que los mapeos se construyen alrededor de declaraciones de clases persistentes y no alrededor de declaraciones de tablas.

Observe que, incluso aunque muchos de los usuarios de Hibernate eligen escribir el XML a mano, existe un número de herramientas para generar el documento de mapeo, incluyendo XDoclet, Middlegen y AndroMDA.

Este es un ejemplo de mapeo:

```
<?xml version="1.0"?>
<!DOCTYPE Hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://Hibernate.sourceforge.net/Hibernate-mapping-3.0.dtd">
<!-- Generated 12/11/2012 11:58:22 AM by Hibernate Tools 3.2.1.GA -->
<Hibernate-mapping>
<class name="modelo.Bien" table="bien" schema="public">
<id name="idbien" type="int">
<column name="idbien" />
<generator class="increment" />
</id>
<property name="txtcodigobien" type="string">
<column name="txtcodigobien" length="40" />
</property>
<property name="dtfecha" type="date">
<column name="dtfecha" length="13" />
</property>
<property name="txtdescripcion" type="string">
<column name="txtdescripcion" length="200" />
</property>
<property name="txtestado" type="string">
<column name="txtestado" length="40" />
</property>
<property name="idproveedor" type="java.lang.Integer">
<column name="idproveedor" />
</property>
<set name="partes" inverse="true">
<key>
<column name="idbien" not-null="true" />
</key>
<one-to-many class="modelo.Parte" />
</set>
<set name="beneficiariobiens" inverse="true">
<key>
<column name="idbien" not-null="true" />
</key>
<one-to-many class="modelo.Beneficiariobien" />
</set>
</class>
</Hibernate-mapping>
```

3.10.2. MAPEO DE HIBERNATE

Este elemento tiene varios atributos opcionales. Los atributos schema y catalog especifican que las tablas a las que se refiere en este mapeo pertenecen al esquema y/o catálogo mencionado(s). De especificarse, los nombres de tablas serán calificados por el nombre del esquema y del catálogo dado. De omitirse, los nombres de las tablas no serán calificados. El atributo default-cascade especifica qué estilo de cascada se debe asumir para las propiedades y colecciones que no especifican un atributo cascade. Por defecto, el atributo auto-import nos permite utilizar nombres de clase sin calificar en el lenguaje de consulta.

```
<Hibernate-mapping
        schema="schemaName"
        catalog="catalogName"
        default-cascade="cascade_style"
        default-access="field|property|ClassName"
        default-lazy="true|false"
        auto-import="true|false"
        package="package.name"
/>
```

- **schema (opcional):** El nombre de un esquema de la base de datos.
- **catalog (opcional):** El nombre de un catálogo de la base de datos.
- **default-cascade (opcional - por defecto es none):** Un estilo de cascada por defecto.
- **default-access (opcional - por defecto es property):** La estrategia que Hibernate debe utilizar para acceder a todas las propiedades. Puede ser una implementación personalizada de PropertyAccessor.
- **default-lazy (opcional - por defecto es true):** El valor por defecto para los atributos lazy no especificados de mapeos de clase y de colección.

- **auto-import (opcional - por defecto es true):** Especifica si podemos utilizar nombres de clases no calificados de clases en este mapeo en el lenguaje de consulta.
- **package (opcional):** Especifica un prefijo de paquete que se debe utilizar para los nombres de clase no calificados en el documento de mapeo.

Si tiene dos clases persistentes con el mismo nombre (sin calificar), debe establecer auto-import="false". Se presentará una excepción si usted intenta asignar dos clases al mismo nombre "importado".

El elemento Hibernate-mapping le permite anidar varios mapeos <class> persistentes, como se mostró anteriormente. Sin embargo, es una buena práctica (y algunas herramientas esperan) que mapee sólamente una clase persistente, o a una sóla jerarquía de clases, en un archivo de mapeo y nombrarlo como la superclase persistente. Por ejemplo, Cat.hbm.xml, Dog.hbm.xml, o si utiliza herencia, Animal.hbm.xml.

3.10.3 CLASE

Puede declarar una clase persistente utilizando el elemento class. Por ejemplo:

```
<class
        name="ClassName"
        table="tableName"
        discriminator-value="discriminator_value"
        mutable="true|false"
        schema="owner"
        catalog="catalog"
        proxy="ProxyInterface"
        dynamic-update="true|false"
        dynamic-insert="true|false"
```

- **name (opcional):** El nombre completamente calificado de la clase Java persistente (o interfaz). Si se omite este atributo, se asume que el mapeo es para una entidad que no es POJO.
- **table (opcional -** por defecto es el nombre de la clase no calificado**)**: El nombre de su tabla en la base de datos.
- **discriminator-value (opcional -** predeterminado al nombre de la clase**)**: Un valor que distingue subclases individuales, usado para el comportamiento polimórfico. Los valores aceptables incluyen null y not null.
- **mutable (opcional, por defecto es true):** Especifica que las instancias de la clase (no) son mutables.
- **schema (opcional):** Sobrescribe el nombre del esquema especificado por el elemento raíz <Hibernate-mapping>.
- **catalog (opcional):** Sobrescribe el nombre del catálogo especificado por el elemento raíz <Hibernate-mapping>.
- **proxy (opcional):** Especifica una interfaz a utilizar para los proxies de inicialización perezosa. Puede especificar el nombre mismo de la clase.
- **dynamic-update (opcional,por defecto es false)**: Especifica que el SQL UPDATE debe ser generado en tiempo de ejecución y puede contener sólamente aquellas columnas cuyos valores hayan cambiado.
- **dynamic-insert (opcional, por defecto es false):** Especifica que el SQL INSERT debe ser generado en tiempo de ejecución y debe contener sólamente aquellas columnas cuyos valores no son nulos.
- **select-before-update (opcional, por defecto es false):** Especifica que Hibernate*nunca* debe realizar un UPDATE SQL a menos de que se tenga certeza de que realmente se haya modificado un objeto. Sólo cuando un objeto transitorio ha sido asociado con una sesión nueva utilizando update(),

Hibernate realizará una SQL SELECT extra para determinar si realmente se necesita un UPDATE.

- **polymorphism (opcional, por defecto es implicit):** Determina si se utiliza polimorfismo de consulta implícito o explícito.
- **where (opcional)** especifica una condición SQL WHERE arbitraria para utilizarla en la recuperación de objetos de esta clase.
- **persister (opcional):** Especifica un ClassPersister personalizado.
- **batch-size (opcional, por defecto es 1)** especifica un "tamaño de lote" para buscar instancias de esta clase por identificador.
- **optimistic-lock (opcional, por defecto es version):** Determina la estrategia optimista de bloqueo.
- **lazy (opcional):** La recuperación perezosa se puede deshabilitar por completo al establecer lazy="false".
- **entity-name (optional - defaults to the class name):** Hibernate3 permite mapear multiples clases.
- **check (opcional):** Una expresión SQL utilizada para generar una restricción *check* multi-filas para la generación automática de esquemas.
- **rowid (opcional):**Hibernate puede utilizar los llamados ROWIDs en las bases de datos. Por ejemplo, en Oracle, Hibernate puede utilizar la columna extra rowid para actualizaciones rápidas si usted establece esta opción como rowid. Un ROWID es un detalle de implementación y representa la posición física de la tupla almacenada.
- **subselect (opcional):** Mapea una entidad inmutable y de sólo lectura a una subselección de base de datos. Es útil si quiere tener una vista en vez de una tabla base. Vea a continuación para obtener más información.
- **abstract (opcional):** Utilizado para marcar superclases abstractas en las jerarquías <union-subclass>.

Es perfectamente aceptable que la clase persistente mencionada sea una interfaz. Puede declarar clases que implementan esa interfaz utilizando el elemento <subclass>. Puede persistir cualquier clase interna *estática*. Debe especificar el nombre de la clase utilizando la forma estándar, por ejemplo, e.g.Foo$Bar.

Las clases inmutables, mutable="false", no pueden ser actualizadas o borradas por la aplicación. Esto le permite a Hibernate realizar ciertas optimizaciones menores de rendimiento.

El atributo opcional proxy activa la inicialización perezosa de instancias persistentes de la clase. Hibernate inicialmente retornará proxies CGLIB que implementan la interfaz mencionada. El objeto persistente real será cargado cuando se invoque un método del proxy.

Por polimorfismo *implícito* se entiende que las instancias de la clase serán devueltas por una consulta que mencione cualquier superclase, o interfaz implementada, o la clase misma; y que las instancias de cualquier subclase de la clase serán retornadas por una petición que nombra a la clase misma. Por polimorfismo *explícito* se entiende que las instancias de la clase serán devueltas sólo por consultas que mencionen explícitamente la clase. Las consultas que mencionen la clase retornarán sólo instancias de subclases mapeadas dentro de esta declaración <class> como una <subclass> o <joined-subclass>. Para la mayoría de los propósitos el valor por defecto, polymorphism="implicit", resulta apropiado. El polimorfismo explícito es útil cuando dos clases diferentes se encuentran mapeadas a la misma tabla. Esto permite tener una clase "liviana" que contenga un subconjunto de columnas de la tabla.

El atributo persister le permite personalizar la estrategia de persistencia para la clase. Por ejemplo, puede especificar su propia subclase de org.Hibernate.persister.EntityPersister, o incluso puede proporcionar una implementación completamente nueva de la interfaz org.Hibernate.persister.ClassPersister que implemente, por ejemplo, la persistencia

por medio de llamadas a procedimientos almacenados, serialización a archivos planos o LDAP.

Los valores de dynamic-update y dynamic-insert no son heredados por las subclases y por lo tanto deben especificarse en los elementos <subclass> o <joined-subclass>. Aunque en algunos casos, estos ajustes pueden incrementar el rendimiento, de hecho en otros casos, podrían disminuirlo.

El uso de select-before-update disminuirá el rendimiento. Es muy útil prevenir que se llame innecesariamente a un disparador de actualización de la base de datos al volver a unir un gráfico de instancias separadas a una Session.

Si activa dynamic-update, usted tendrá la opción de estrategias de bloqueo optimistas:

- version: chequea las columnas de versión/sello de fecha
- all: chequea todas las columnas
- dirty: chequea las columnas modificadas permitiendo algunas actualizaciones concurrentes
- none: no utilice bloqueo optimista

Le recomendamos mucho que utilice columnas de versión/sello de fecha para el bloqueo optimista con Hibernate. Esta estrategia optimiza el rendimiento y maneja correctamente las modificaciones realizadas a las instancias separadas, (por ejemplo, cuando se utiliza Session.merge().

Para un mapeo de Hibernate, no hay diferencia entre una vista y una tabla base. Esto es transparente a nivel de base de datos, aunque algunos DBMS no soportan correctamente las vistas, especialmente con las actualizaciones. A veces usted quiere utilizar una vista, pero no puede crear una en la base de datos (por ejemplo, con un esquema heredado). En este caso, usted puede mapear una entidad inmutable de sólo lectura a una expresión de subconsulta SQL dada.

```
<class name="Summary">
   <subselect>
      select item.name, max(bid.amount), count(*)
      from item
      join bid on bid.item_id = item.id
      group by item.name
   </subselect>
   <synchronize table="item"/>
   <synchronize table="bid"/>
   <id name="name"/>
   ...
</class>
```

3.10.4. ID

Las clases mapeadas *tienen* que declarar la columna de clave primaria de la tabla de la base de datos. La mayoría de las clases también tendrán una propiedad de estilo Javabeans que tenga el identificador único de una instancia. El elemento <id> define el mapeo de esa propiedad a la columna de clave primaria.

```
<id
      name="propertyName"
      type="typename"
      column="column_name"
      unsaved-value="null|any|none|undefined|id_value"
      access="field|property|ClassName">
      node="element-name|@attribute-name|element/@attribute|."

<generator class="generatorClass"/>
</id>
```

- **name (opcional):** El nombre de la propiedad del identificador.
- **type (opcional):** un nombre que indica el tipo de Hibernate.
- **column (opcional -** por defecto es el nombre de la propiedad): El nombre de la columna de la clave principal.
- **unsaved-value (opcional - por defecto es un valor "sensible"):** Un valor de la propiedad identificadora que indica que una instancia está recién instanciada (sin

guardar), distinguiéndola de las instancias separadas que fueron guardadas o cargadas en una sesión previa.

- **access (opcional - por defecto es property):** La estrategia que Hibernate debe utilizar para acceder al valor de la propiedad.

Si se omite el atributo name, se asume que la clase no tiene propiedad identificadora.

El atributo unsaved-value casi nunca se necesita en Hibernate3.

Hay una declaración <composite-id> opcional para permitir acceso a los datos heredados con claves compuestas. Le disuadimos seriamente de su utilización para cualquier otra cosa.

- **Generador**

El elemento hijo opcional <generator> nombra una clase Java utilizada para generar identificadores únicos para instancias de la clase persistente. De requerirse algún parámetro para configurar o inicializar la instancia del generador, se pasa utilizando el elemento <param>.

```
<id name="id" type="long" column="cat_id">
      <generator class="org.Hibernate.id.TableHiLoGenerator">
            <param name="table"
>uid_table</param>
            <param name="column"
>next_hi_value_column</param>
      </generator>
</id
>
```

Todos los generadores implementan la interfaz org.Hibernate.id.IdentifierGenerator. Esta es una interfaz muy simple. Algunas aplicaciones pueden decidir brindar sus propias implementaciones especializadas. Sin embargo, Hibernate provee un rango de implementaciones ya incorporadas. Los nombres de atajo para los generadores incorporados son los siguientes:

- **increment**

Genera indentificadores de tipo long, short o int que sólamente son únicos cuando ningún otro proceso está insertando datos en la misma tabla. *No lo utilice en un clúster.*

- **identity**

Soporta columnas de identidad en DB2, MySQL, MS SQL Server, Sybase y HypersonicSQL. El identificador devuelto es de tipo long, short o int.

- **sequence**

Usa una secuencia en DB2, PostgreSQL, Oracle, SAP DB, McKoi o un generador en Interbase. El identificador devuelto es de tipo long, short o int.

- **hilo**

Utiliza un algoritmo alto/bajo para generar eficientemente identificadores de tipo long, short o int, dada una tabla y columna como fuente de valores altos (por defecto Hibernate_unique_key y next_hi respectivamente). El algoritmo alto/bajo genera identificadores que son únicos sólamente para una base de datos particular.

- **seqhilo**

Utiliza un algoritmo alto/bajo para generar eficientemente identificadores de tipo long, short o int, dada una secuencia de base de datos.

- **uuid**

Utiliza un algoritmo UUID de 128 bits para generar identificadores de tipo cadena, únicos dentro de una red (se utiliza la direccón IP). El UUID se codifica como una cadena hexadecimal de 32 dígitos de largo.

- **assigned**

Deja a la aplicación asignar un identificador al objeto antes de que se llame a save(). Esta es la estrategia por defecto si no se especifica un elemento <generator>.

- **select**

Recupera una clave principal asignada por un disparador de base de datos seleccionando la fila por alguna clave única y recuperando el valor de la clave principal.

- **foreign**

Utiliza el identificador de otro objeto asociado. Generalmente se usa en conjunto cón a una asociación de clave principal <one-to-one>.

- **sequence-identity**

Una estrategia de generación de secuencias especilizadas que utiliza una secuencia de base de datos para el valor real de la generación, pero combina esto junto con JDBC3 getGeneratedKeys para devolver el valor del identificador generado como parte de la ejecución de la declaración de inserción. Esta estrategia está soportada sólamente en los controladores 10g de Oracle destinados para JDK1.4. Los comentarios en estas declaraciones de inserción están desactivados debido a un error en los controladores de Oracle.

- **Columnas de identidad y secuencias**

Para las bases de datos que soportan columnas de identidad (DB2, MySQL, Sybase, MS SQL), puede utilizar generación de claves identity. Para las bases de datos que soportan las secuencias (DB2, Oracle, PostgreSQL, Interbase, McKoi, SAP DB) puede utilizar la generación de claves del estilo sequence. Ambas estrategias requieren dos consultas SQL para insertar un nuevo objeto. Por ejemplo:

```
<id name="id" type="long" column="person_id">
      <generator class="sequence">
            <param name="sequence"
>person_id_sequence</param>
      </generator>
</id>

<id name="id" type="long" column="person_id" unsaved-value="0">
      <generator class="identity"/>
</id>
```

Para desarrollos a través de plataformas, la estrategia native eligirá entre las estrategias identity, sequence e hilo, dependiendo de las capacidades de la base de datos subyacente.

- **Claves primarias asignadas por disparadores**

Hibernate no genera DDL con disparadores. Es para los esquemas heredados solamente.

En el ejemplo anterior, hay una propiedad única llamada socialSecurityNumber, Esta está definida por la clase, como una clave natural y una clave sustituta llamada person_id, cuyo valor es generado por un disparador.

```
<id name="id" type="long" column="person_id">
      <generator class="select">
            <param name="key"
>socialSecurityNumber</param>
      </generator>
</id>
```

- **composite-id**

```
<composite-id
      name="propertyName"
      class="ClassName"
      mapped="true|false"
      access="field|property|ClassName">
      node="element-name|."
      <key-
property name="propertyName" type="typename" column="column_name"/>
      <key-many-to-
one name="propertyName" class="ClassName" column="column_name"/>
      ……
</composite-id>
```

Una tabla con clave compuesta se puede mapear con múltiples propiedades de la clase como propiedades identificadoras. El elemento <composite-id> acepta los mapeos de propiedad <key-property> y los mapeos <key-many-to-one> como elementos hijos.

```
<composite-id>
      <key-property name="medicareNumber"/>
      <key-property name="dependent"/>
</composite-id>
```

La clase persistente *tiene* que sobrescribir equals() y hashCode() para implementar la igualdad del identificador compuesto. También tiene que implementar Serializable.

Desafortunadamente, este enfoque significa que un objeto persistente es su propio identificador.Debe instanciar una instancia de la clase persistente y poblar sus propiedades identificadoras antes de que pueda load() el estado persistente asociado a una clave compuesta. Este enfoque lo denominamos un identificador compuesto *incluído* y no lo recomendamos para aplicaciones serias.

Un segundo enfoque es lo que denominamos un identificador compuesto *mapeado*, en donde las propiedades del identificador nombradas dentro del elemento <composite-

id> son duplicadas tanto en la clase persistente como en la clase identificadora separada.

```
<composite-id class="MedicareId" mapped="true">
      <key-property name="medicareNumber"/>
      <key-property name="dependent"/>
</composite-id>
```

En este ejemplo, tanto la clase identificadora compuesta MedicareId como la clase de entidad misma tienen propiedades denominadas medicareNumber y dependent. La clase identificadora tiene que sobrescribir equals() y hashCode() e implementar Serializable. La desventaja principal de este enfoque es la duplicación de código.

Los siguientes atributos se utilizan para especificar un identificador compuesto mapeado:

- **mapped (opcional, por defecto es false):** indica que se utiliza un identificador compuesto mapeado y que los mapeos de propiedad contenidos se refieren tanto a la clase de entidad como a la clase identificadora compuesta.
- **class (opcional, pero requerida por un identificador compuesto mapeado):** La clase se utiliza como un identificador compuesto.
- **name (opcional, se necesita para este enfoque):** Una propiedad de tipo componente que tiene el identificador compuesto.
- **access (opcional - por defecto es property):** La estrategia que Hibernate utiliza para acceder al valor de la propiedad.
- **class (opcional - por defecto es el tipo de propiedad determinado por la reflección):** la clase componente utilizada como un identificador compuesto.

Este tercer enfoque, un *componente identificador* es el que recomendamos para casi todas las aplicaciones.

3.10.7. DISCRIMINADOR

Se necesita el elemento <discriminator> para la persistencia polimórfica utilizando la estrategia de mapeo de tabla-por-jerarquía-de-clases. Declara una columna discriminadora de la tabla. La columna discriminidora contiene valores de marca que le dicen a la capa de persistencia qué subclase instanciar para una fila en particular. Se puede utilizar un conjunto restringido de tipos: string, character, integer, byte, short, boolean, yes_no, true_false.

```
<discriminator
      column="discriminator_column"
      type="discriminator_type"
      force="true|false"
      insert="true|false"
      formula="arbitrary sql expression"
/>
```

- **column (opcional - por defecto es class)** el nombre de la columna discriminadora.
- **type (opcional - por defecto es string)** un nombre que indica el tipo Hibernate.
- **force (opcional - por defecto es false)** "fuerza" a Hibernate para especificar los valores discriminadores permitidos incluso cuando se recuperan todas las instancias de la clase raíz.
- **insert (opcional - por defecto es true):** establecido como false si su columna discriminadora también es parte de un identificador mapeado compuesto. Lle dice a Hibernate que no incluya la columna en los SQLs INSERT.
- **formula (opcional):** una expresión SQL arbitraria que se ejecuta cuando se tenga que evaluar un tipo. Permite la discriminación con base en el contenido.

- Los valores reales de la columna discriminadora están especificados por el atributo discriminator-value de los elementos <class> y <subclass>.

El atributo force es sólamente útil si la tabla contiene filas con valores discriminadores "extra" que no estén mapeados a una clase persistente. Generalmente este no es el caso.

El atributo formula le permite declarar una expresión SQL arbitraria que será utilizada para evaluar el tipo de una fila. Por ejemplo:

```
<discriminator
    formula="case when CLASS_TYPE in ('a', 'b', 'c') then 0 else 1
  end"
    type="integer"/>
```

3.10.8. VERSIÓN (OPCIONAL)

El elemento <version> es opcional e indica que la tabla contiene datos versionados. Esto es particularmente útil si planea utilizar transacciones largas. Vea a continuación para obtener mayor información:

```
<version
        column="version_column"
        name="propertyName"
        type="typename"
        access="field|property|ClassName"
        unsaved-value="null|negative|undefined"
        generated="never|always"
        insert="true|false"
        node="element-name|@attribute-name|element/@attribute|."
/>
```

- **column (opcional -** por defecto es el nombre de la propiedad): El nombre de la columna que tiene el número de la versión.
- **name:** El nombre de una propiedad de la clase persistente.
- **type (opcional - por defecto es integer):** El tipo del número de la versión.

- **access (opcional - por defecto es property):** La estrategia que Hibernate utiliza para acceder al valor de la propiedad.
- **unsaved-value (opcional - por defecto es undefined):** Un valor de la propiedad de versión que indica que una instancia se encuentra recién instanciada (sin guardar), distinguiéndola de las instancias separadas que se guardaron o se cargaron en una sesión previa. Undefined especifica que se debe utilizar el valor de la propiedad identificadora.
- **generated (opcional - por defecto es never):** Especifica que este valor de la propiedad de la versión es generado por la base de datos. Vea la discusión de las propiedades generadas para obtener mayor información.
- **insert (opcional - por defectos es true):** Especifica si la columna de la versión debe incluirse en las declaraciones de inserción SQL. Se puede configurar como false si la columna de la base de datos se define con un valor predeterminado de 0.

Los números de versión pueden ser de tipo Hibernate long, integer, short, timestamp o calendar.

Una propiedad de versión o de sello de fecha nunca debe ser nula para una instancia separada. Hibernate detectará cualquier instancia con una versión o sello de fecha nulo como transitoria, sin importar qué otras estrategias unsaved-value se hayan especificado. El declarar una propiedad de versión o sello de fecha nulable es una forma fácil de evitar cualquier problema con la re-unión transitiva en Hibernate. Es especialmente útil para la gente que utiliza identificadores asignados o claves compuestas.

3.10.9. TIMESTAMP (OPCIONAL)

El elemento opcional <timestamp> indica que la tabla contiene datos con sellos de fecha. Esto brinda una alternativa al versionado. Los sellos de tiempo (timestamps) son por naturaleza una implementación menos segura del bloqueo optimista. Sin embargo, a veces la aplicación puede usar los sellos de fecha de otras maneras.

```
<timestamp
      column="timestamp_column"
      name="propertyName"
      access="field|property|ClassName"
      unsaved-value="null|undefined"
      source="vm|db"
      generated="never|always"
      node="element-name|@attribute-name|element/@attribute|."
/>
```

- **column (opcional -** por defecto es el nombre de la propiedad): El nombre de una columna que tiene el sello de fecha.
- **name:** El nombre de una propiedad del estilo JavaBeans de tipo Java Date o Timestamp de la clase persistente.
- **access (opcional - por defecto es property):** La estrategia que Hibernate utiliza para acceder al valor de la propiedad.
- **unsaved-value (opcional - por defecto es null):** Un valor de propiedad de versión que indica que una instancia está recién instanciada (sin guardar), distinguiéndola de instancias separadas que hayan sido guardadas o cargadas en una sesión previa. Undefined especifica que debe utilizarse el valor de la propiedad identificadora.
- **source (opcional - por defecto es vm):** Los sellos de fecha con base en la base de datos provocan un gasto general debido a que Hibernate tiene que

llegar hasta la base de datos para poder determinar el "siguiente valor". Es más seguro utilizarlo en entornos con clústers. No todos los Dialects soportan la recuperación del sello de fecha actual de la base de datos. Los otros pueden ser poco seguros para utilizarlos como bloqueo debido a la falta de precisión (por ejemplo, Oracle 8).

- **generated (opcional - por defecto es never):** Especifica que este valor de la propiedad del sello de fecha en realidad es generado por la base de datos.

3.10.10 PROPERTY

El elemento <property> declara una propiedad persistente estilo JavaBean de la clase.

```
<property
      name="propertyName"
      column="column_name"
      type="typename"
      update="true|false"
      insert="true|false"
      formula="arbitrary SQL expression"
      access="field|property|ClassName"
      lazy="true|false"
      unique="true|false"
      not-null="true|false"
      optimistic-lock="true|false"
      generated="never|insert|always"
      node="element-name|@attribute-name|element/@attribute|."
      index="index_name"
      unique_key="unique_key_id"
      length="L"
      precision="P"
      scale="S"
/>
```

- **name:** el nombre de la propiedad, con la letra inicial en minúscula.

- **column (opcional - por defecto es el nombre de la propiedad):** El nombre de la columna de la tabla de base de datos mapeada. Esto se puede especificar también con los elemento(s) anidado(s) <column>.
- **type (opcional):** un nombre que indica el tipo de Hibernate.
- **update, insert (opcional - por defecto es true):** Especifica que las columnas mapeadas deben ser incluídas en las declaraciones SQL UPDATE y/o INSERT. Especificando ambas como false permite una propiedad "derivada", cuyo valor se inicia desde alguna otra propiedad que mapee a la misma columna (o columnas) o por un disparador u otra aplicación.
- **formula (opcional):** una expresión SQL que define el valor para una propiedad *computada*. Las propiedades computadas no tienen una columna mapeada propia.
- **access (opcional - por defecto es property):** La estrategia que Hibernate utiliza para acceder al valor de la propiedad.
- **lazy (opcional - por defecto es false):** Especifica que se debe recuperar perezosamente esta propiedad cuando se acceda por primera vez la variable de instancia. Requiere instrumentación de código byte en tiempo de compilación.
- **unique (opcional):** Activa la generación DDL de una restricción de unicidad para las columnas. Además, permite que ésta sea el objetivo de una property-ref.
- **not-null (opcional):** Activa la generación DDL de una restricción de nulabilidad para las columnas.
- **optimistic-lock (opcional - por defecto es true):** Específica que las actualizaciones a esta propiedad requieren o no de la obtención de un bloqueooptimista. En otras palabras, determina si debe ocurrir un incremento de versión cuando la propiedad se encuentre desactualizada.
- **generated (opcional - por defecto es never):** Especifica que este valor de lapropiedad es de hecho generado por la base de datos. Consulte discusión sobre las propiedades generadas para obtener mayor información.

Si no especifica un tipo, Hibernate utilizará reflección sobre la propiedad mencionada para deducir el tipo Hibernate correcto. Hibernate intentará interpretar el nombre de la clase de retorno del getter de la propiedad utilizando las reglas 2, 3 y 4 en ese mismo orden. En algunos casos necesitará el atributo type. Por ejemplo, para distinguir entre Hibernate.DATE y Hibernate.TIMESTAMP, o especificar un tipo personalizado.

El atributo access le permite controlar el cómo Hibernate accederá a la propiedad en tiempo de ejecución. Por defecto, Hibernate llamará al par de getter/setter de la propiedad. Si usted especifica access="field", Hibernate se saltará el par get/set y accederá al campo directamente utilizando reflección. Puede especificar su propia estrategia de acceso a la propiedad mencionando una clase que implemente la interfaz org.Hibernate.property.PropertyAccessor.

Una funcionalidad especialmente poderosa son las propiedades derivadas. Estas propiedades son, por definición, de sólo lectura. El valor de la propiedad se computa en tiempo de carga. Usted declara la computación como una expresión SQL y ésta se traduce como una cláusula de subconsulta SELECT en la consulta SQL que carga una instancia:

```
<property name="totalPrice"
  formula="( SELECT SUM (li.quantity*p.price) FROM LineItem li, Product p
        WHERE li.productId = p.productId
        AND li.customerId = customerId
        AND li.orderNumber = orderNumber )"/>
```

Puede referenciar la tabla de las entidades sin declarar un alias o una columna particular. En el ejemplo dado sería customerId. También puede utilizar el elemento anidado de mapeo <formula> si no quiere utilizar el atributo.

3.10.11. RELACION MANY-TO-ONE

Una asociación ordinaria a otra clase persistente se declara utilizando el elemento many-to-one. El modelo relacional es una asociación muchos-a-uno; una clave foránea en una tabla referencia la columna (o columnas) de la clave principal de la tabla destino.

```
<many-to-one
      name="propertyName"
      column="column_name"
      class="ClassName"
      cascade="cascade_style"
      fetch="join|select"
      update="true|false"
      insert="true|false"
      property-ref="propertyNameFromAssociatedClass"
      access="field|property|ClassName"
      unique="true|false"
      not-null="true|false"
      optimistic-lock="true|false"
lazy="proxy|no-proxy|false"
not-found="ignore|exception"
      entity-name="EntityName"
      formula="arbitrary SQL expression"
      node="element-name|@attribute-name|element/@attribute|."
      embed-xml="true|false"
      index="index_name"
      unique_key="unique_key_id"
      foreign-key="foreign_key_name"
/>
```

Este es un ejemplo de una declaración típica muchos-a-uno:

```
<many-to-one name="product" class="Product" column="PRODUCT_ID"/>
```

3.10.12. ONE-TO-ONE

Una asociación uno-a-uno (one-to-one) a otra clase persistente se declara utilizando un elemento one-to-one.

```
<one-to-one
      name="propertyName"
      class="ClassName"
      cascade="cascade_style"
      constrained="true|false"
      fetch="join|select"
      property-ref="propertyNameFromAssociatedClass"
      access="field|property|ClassName"
      formula="any SQL expression"
lazy="proxy|no-proxy|false"
entity-name="EntityName"
      node="element-name|@attribute-name|element/@attribute|."
      embed-xml="true|false"
      foreign-key="foreign_key_name"/>
```

Existen dos variedades de asociaciones uno-a-uno:

- asociaciones de clave primaria
- asociaciones de clave foránea única

Las asociaciones de claves principales no necesitan una columna extra de la tabla. Si dos filas están relacionadas por la asociación entonces las dos filas de tablas comparten el mismo valor de clave principal. Para que dos objetos estén relacionados por una asociación de clave principal, asegúrese de que se les asigne el mismo valor de identificador.

Para una asociación de clave principal, agregue los siguientes mapeos a Employee y Person respectivamente:

Asegúrese de que las claves principales de las filas relacionadas en las tablas PERSON y EMPLOYEE sean iguales. Utilizamos una estrategia especial de generación de identificador de Hibernate denominada foreign.

A una instancia recién guardada de Person se le asigna el mismo valor de clave principal que se le asignó a la instancia Employee referida por la propiedad employee de esa Person.

Opcionalmente, una clave foránea con una restricción de unicidad, desde Employee a Person, se puede expresar como:

```
<many-to-one name="person" class="Person" column="PERSON_ID" unique="true"/>
```

Esta asociación puede hacerse bidireccional agregando lo siguiente al mapeo de Person:

```
<one-to-one name="employee" class="Employee" property-ref="person"/>
```

3.10.13. COMPONENTE Y COMPONENTE DINÁMICO

El elemento <component> mapea propiedades de un objeto hijo a columnas de la tabla de la clase padre. Los componentes pueden, a su vez, declarar sus propias propiedades, componentes o colecciones. Vea a continuación los "componentes":

```
<component
name="propertyName"
class="className"
insert="true|false"
update="true|false"
access="field|property|ClassName"
lazy="true|false"
```

Las etiquetas hijas <property> mapean propiedades de la clase hija a las columnas de la tabla.

El elemento <component> permite un subelemento <parent> que mapea una propiedad de la clase del componente como una referencia a la entidad contenedora.

3.10.14. SUBCLASE

La persistencia polimórfica requiere la declaración de cada subclase de la clase persistente raíz. Para la estrategia de mapeo tabla-por-jerarquía-de-clases, se utiliza la declaración <subclass>. Por ejemplo:

```
<subclass
      name="ClassName"
      discriminator-value="discriminator_value"
      proxy="ProxyInterface"
      lazy="true|false"
      dynamic-update="true|false"
      dynamic-insert="true|false"
      entity-name="EntityName"
```

- **name:** El nombre de clase completamente calificado de la subclase.
- **discriminator-value (opcional - por defecto es el nombre de la clase):** Un valorque distingue subclases individuales.
- **proxy (opcional):** Especifica una clase o interfaz que se utiliza para proxies de inicialización perezosa.
- **lazy (opcional, por defecto es true):** El establecer lazy="false" desactiva el uso de la recuperación perezosa.

Cada subclase debe declarar sus propias propiedades persistentes y subclases. Se asume que las propiedades <version> y <id> son heredadas de la clase raíz. Cada subclase en una jerarquía tiene que definir un discriminator-value único. Si no se especifica ninguno entonces se utiliza el nombre completamente calificado de clase Java.

3.10.15. JOIN

Al utilizar el elemento <join>, es posible mapear las propiedades de una clase a varias tablas que tengan una relación uno-a-uno. Por ejemplo:

```
<join
      table="tablename"
      schema="owner"
      catalog="catalog"
      fetch="join|select"
      inverse="true|false"
      optional="true|false">
<key ... />

<property ... />
      ...
</join>
```

- **table:** El nombre de la tabla unida.
- **schema (opcional):** Sobrescribe el nombre del esquema especificado por el elemento raíz <Hibernate-mapping>.
- **catalog (opcional):** Sobrescribe el nombre del catálogo especificado por el elemento raíz <Hibernate-mapping>.
- **fetch (opcional - por defecto es join):** Si se establece como join, por defecto, Hibernate utilizará una unión interior (inner join) para recuperar un <join> definido por una clase o sus superclases. Utilizará una unión externa (outer join) para un <join> definido por una subclase. Si se establece como select, entonces Hibernate utilizará una selección secuencial para un <join> definido en una subclase. Esto se publicará sólamente si una fila representa una instanciade la subclase. Las uniones interiores todavía serán utilizadas pararecuperar un <join> definido por la clase y sus superclases.
- **inverse (opcional - por defecto es false):** De activarse, Hibernate no tratará de insertar o actualizar las propiedades definidas por esta unión.
- **optional (opcional - por defecto es false):** De activarse, Hibernate insertará una fila sólo si las propiedades definidas por esta unión son no-nulas.

  Siempre utilizará una unión externa para recuperar las propiedades.

Por ejemplo, la información domiciliaria de una persona se puede mapear a una tabla separada, preservando a la vez la semántica de tipo de valor para todas las propiedades:

```
<class name="Person"
    table="PERSON">

    <id name="id" column="PERSON_ID"
>...</id>

    <join table="ADDRESS">
        <key column="ADDRESS_ID"/>
        <property name="address"/>
        <property name="zip"/>
        <property name="country"/>
    </join>
    ...
```

Con frecuencia, esta funcionalidad sólamente es útil para los modelos de datos heredados. Recomendamos menos tablas que clases y un modelo de dominio más detallado. Sin embargo, es útil para cambiar entre estrategias de mapeo de herencias en una misma jerarquía.

3.11 TIPOS DE DATOS HIBERNATE

3.11.1. ENTIDADES Y VALORES

En relación con el servicio de persistencia, los objetos a nivel de lenguaje Java se clasifican en dos grupos:

Una entidad existe independientemente de cualquier otro objeto que referencie a la entidad. Compare esto con el modelo habitual de Java en donde un objeto no referenciado es recolectado como basura. Las entidades deben ser guardadas y borradas explícitamente. Sin embargo, los grabados y borrados se pueden tratar en cascada desde una entidad padre a sus hijos. Esto es diferente al modelo de

persistencia de objetos por alcance (ODMG) y corresponde más a cómo se utilizan habitualmente los objetos de aplicación en sistemas grandes. Las entidades soportan referencias circulares y compartidas, que también pueden ser versionadas.

El estado persistente de una entidad consta de las referencias a otras entidades e instancias de tipo *valor*. Los valores son primitivos: colecciones (no lo que está dentro de la colección), componentes y ciertos objetos inmutables. A diferencia de las entidades, los valores en particular las colecciones y los componentes, *son* persistidos y borrados por alcance. Como los objetos valor y primitivos son persistidos y borrados junto con sus entidades contenedoras, no se pueden versionar independientemente. Los valores no tienen identidad independiente, por lo que dos entidades o colleciones no los pueden compartir.

Hasta ahora, hemos estado utilizando el término "clase persistente" para referirnos a entidades. Continuaremos haciéndolo así. Sin embargo, no todas la clases con estado persistente definidas por el usuario son entidades. Un *componente* es una clase definida por el usuario con semántica de valor. Una propiedad Java de tipo java.lang.String también tiene semántica de valor. Dada esta definición, podemos decir que todos los tipo (clases) provistos por el JDK tienen una semántica de tipo valor en Java, mientras que los tipos definidos por el usuario se pueden mapear con semántica de tipo valor o de entidad. La desición corre por cuenta del desarrollador de la aplicación. Una clase entidad en un modelo de dominio son las referencias compartidas a una sola instancia de esa clase, mientras que la composición o agregación usualmente se traducen a un tipo de valor.

EL desafío es mapear el sistema de tipos de Java (la definición de entidades y tipos de valor de los desarrolladores al sistema de tipos de SQL/la base de datos. El puente entre ambos sistemas lo brinda Hibernate. Para las entidades utilizamos <class>, <subclass>, etc. Para los tipos de valor utilizamos <property>, <component>, etc, usualmente con un atributo type. El valor de este atributo es el nombre de un tipo de

mapeo de Hibernate. Hibernate proporciona un rango de mapeos para tipos de valores del JDK estándar. Puede escribir sus propios mapeos de tipo e implementar sus estrategias de conversión personalizadas.

Todos los tipos incorporados de Hibernate soportan la semántica de nulos, a excepción de las colecciones.

3.11.2. TIPOS DE VALORES BÁSICOS

Los tipos de mapeo básicos incorporados se pueden categorizar así:

Integer, Long, Short, Float, Double, Character, Byte, Boolean, Yes_No, True_False

Mapeos de tipos de primitivos de Java o de clases de envoltura a los tipos de columna SQL (específica del vendedor). Boolean, yes_no y true_false son codificaciones alternativas a boolean de Java o java.lang.Boolean.

String: Un mapeo del tipo java.lang.String a VARCHAR (u Oracle VAARCHAR2).

date, time, timestamp: Mapeos de tipo desde java.util.Date y sus subclases a tipos SQL DATE, TIME y TIMESTAMP (o equivalente).

calendar, calendar_date : Mapeos de tipo desde java.util.Date y tipos SQL TIMESTAMP y DATE (o equivalente).

big_decimal, big_integer: Mapeos de tipo desde java.math.BigDecimal y java.math.BigInteger a NUMERIC (o NUMBER de Oracle).

locale, timezone, currency: Mapeos de tipo desde java.util.Locale, java.util.TimeZone y java.util.Currency a VARCHAR (o VARCHAR2 de Oracle). Las instancias de Locale y Currency son mapeadas a sus códigos ISO. Las instancias de TimeZone son mapeadas a sus ID.

Class: Un mapeo de tipo java.lang.Class a VARCHAR (o VARCHAR2 de Oracle). Una Class es mapeada a su nombre completamente calificado.

Binary: Mapea arreglos de bytes a un tipo binario SQL apropiado.

Text: Mapea cadenas largas de Java al tipo SQL CLOB o TEXT.

Serializable: Mapea tipos serializables Java a un tipo binario SQL apropiado. También puede indicar el tipo serializable de Hibernate con el nombre de una clase o interfaz serializable Java que no sea por defecto un tipo básico.

Los identificadores únicos de entidades y colecciones pueden ser de cualquier tipo básico excepto binary, blob y clob. Los identificadores compuestos también están permitidos, a continuación encontrará mayor información.

Los tipos de valor básicos tienen sus constantes Type correspondientes definidas en org.Hibernate.Hibernate. Por ejemplo, Hibernate.STRING representa el tipo string.

3.11.3 TIPOS DE VALOR PERSONALIZADOS

Es relativamente fácil para los desarrolladores crear sus propios tipos de valor. Por ejemplo, puede que quiera persistir propiedades del tipo java.lang.BigInteger a columnas VARCHAR. Hibernate no provee un tipo incorporado para esto. Los tipos personalizados no están limitados a mapear una propiedad o elemento de colección a una sola columna de tabla. Así, por ejemplo, podría tener una propiedad Java getName()/setName() de tipo java.lang.String que es persistida a las columnas FIRST_NAME, INITIAL, SURNAME.

Para implementar un tipo personalizado, implemente org.Hibernate.UserType o org.Hibernate.CompositeUserType y declare las propiedades utilizando el nombre de clase completamente calificado del tipo. Revise org.Hibernate.test.DoubleStringType para ver qué clases de cosas son posibles.

```
<property name="twoStrings" type="org.Hibernate.test.DoubleStringType">
   <column name="first_string"/>
   <column name="second_string"/>
</property>
```

3.12. MAPEO DE UNA CLASE MÁS DE UNA VEZ

Es posible proporcionar más de un mapeo para una clase persistente en particular. En este caso usted debe especificar un nombre de entidad para aclarar entre las instancias de las dos entidades mapeadas. Por defecto, el nombre de la entidad es el mismo que el nombre de la clase. Las asociaciones ahora se especifican utilizando entity-name en lugar de class.

```
<class name="Contract" table="Contracts"entity-name="CurrentContract">
<set name="history" inverse="true"order-by="effectiveEndDate desc">
<key column="currentContractId"/>
<one-to-many entity-name="HistoricalContract"/>
</set>
</class>
<class name="Contract" table="ContractHistory"entity-name="HistoricalContract">
<many-to-one name="currentContract"column="currentContractId"entity-
name="CurrentContract"/>
</class>
```

3.13. IDENTIFICADORES SQL EN COMILLAS

Puede forzar a Hibernate a que utilice comillas con un identificador en el SQL generado encerrando el nombre de tabla o de columna entre comillas sencillas en el documento de mapeo. Hibernate utilizará el estilo de comillas para el Dialect SQL. Usualmente comillas dobles, a excepción de corchetes para SQL Server y comillas sencillas para MySQL.

```
<class name="LineItem" table="`Line Item`">
   <id name="id" column="`Item Id`"/><generator class="assigned"/></id>
   <property name="itemNumber" column="`Item #`"/>
  </class>
```

3.14. ALTERNATIVAS DE METADATOS

XML no es para todo el mundo, así que hay algunas formas opcionales de definir metadatos de mapeo O/R en Hibernate.

3.14.1. UTLIZACIÓN DE ANOTACIONES JDK 5.0

JDK 5.0 introdujo anotaciones del estilo XDoclet a nivel del lenguaje con chequeo seguro de tipos en tiempo de compilación. Este mecanismo es más potente que las anotaciones XDoclet y es mejor soportado por herramientas e IDEs. IntelliJ IDEA, por ejemplo, soporta auto-completación además de resalte de sintaxis de las anotaciones JDK 5.0. La nueva revisión de la especificación de EJB (JSR-220) utiliza anotaciones JDK 5.0 como el mecanismo principal de metadatos para beans de entidad. Hibernate3 implementa el EntityManager del JSR-220 (la API de persistencia). El soporte para metadatos de mapeo está disponible por medio del paquete Anotaciones de Hibernate, como una descarga separada. Tanto los metadatos de EJB3 (JSR-220) como de Hibernate3 se encuentran soportados.

Este es un ejemplo de una clase POJO anotada como un bean de entidad EJB:

```
@Entity(access = AccessType.FIELD)
public class Customer implements Serializable {
    @Id;
    Long id;
    String firstName;
    String lastName;
    Date birthday;
    @Transient
    Integer age;
    @Embedded
    private Address homeAddress;
   @OneToMany(cascade=CascadeType.ALL)
    @JoinColumn(name="CUSTOMER_ID")
    Set<Order
> orders;
    // Getter/setter and business methods
}
```

3.15. PROPIEDADES GENERADAS

Las propiedades generadas son propiedades cuyos valores son generados por la base de datos. Usualmente, las aplicaciones de Hibernate necesitan refrescar los objetos que contenian cualquier propiedad para la cual la base de datos generará valores. Sin embargo, el marcar propiedades como generadas deja que la aplicación delegue esta responsabilidad a Hibernate. Cuando Hibernate emite un INSERT or UPDATE SQL para una entidad la cual ha definido propiedades generadas, inmediatamente emite un select para recuperar los valores generados.

Las propiedades marcadas como generadas tienen que ser además no insertables y no actualizables. Sólamente las versiones, sellos de fecha, y propiedades simples se pueden marcar como generadas.

never (por defecto): el valor dado de la propiedad no es generado dentro de la base de datos.

insert: el valor dado de la propiedad es generado en insert, pero no es regenerado en las actualizaciones posteriores. Las propiedades como fecha-creada (created-date) se encuentran dentro de esta categoría. Aunque las propiedades versión y sello de fecha se pueden marcar como generadas, esta opción no se encuentra disponible.

always: el valor de la propiedad es generado tanto en insert como en update.

3.16. TRANSACCIONES Y CONCURRENCIAS

El punto más importante sobre Hibernate y el control de concurrencia es que es fácil de comprender. Hibernate usa directamente conexiones JDBC y recursos JTA sin agregar ningún comportamiento de bloqueo adicional.

Hibernate no bloquea objetos en la memoria. Su aplicación puede esperar el comportamiento definido por el nivel de aislamiento de sus transacciones de las bases de datos. Gracias a la Session, la cual también es un caché con alcance de

transacción, Hibernate proporciona lecturas repetidas para búsquedas del identificador y consultas de entidad y no consultas de reporte que retornan valores escalares.

Además del versionado del control de concurrencia optimista automático, Hibernate también ofrece una API (menor) para bloqueo pesimista de filas, usando la sintáxis SELECT FOR UPDATE. Esta API y el control de concurrencia optimista se discuten más adelante en este capítulo.

3.16.1. ÁMBITOS DE SESIÓN Y DE TRANSACCIÓN

Una SessionFactory es un objeto seguro entre hilos y costoso de crear pensado para que todas las hebras de la aplicación lo compartan. Se crea una sola vez, usualmente en el inicio de la aplicación, a partir de una instancia Configuration.

Una Session es un objeto de bajo costo, inseguro entre hilos que se debe utilizar una sola vez y luego se debe descartar: para un sólo pedido, una sola conversación o una sóla unidad de trabajo. Una Session no obtendrá una Connection JDBC o un Datasource a menos de que sea necesario. No consumirá recursos hasta que se utilice.

Una transacción de la base de datos tiene que ser tan corta como sea posible para reducir la contención de bloqueos en la base de datos. Las transacciones largas de la base de datos prevendrán a su aplicación de escalar a una carga altamente concurrente. Por lo tanto, no se recomienda que mantenga una transacción de la base de datos abierta durante el tiempo para pensar del usuario, hasta que la unidad de trabajo se encuentre completa.

¿Cuál es el ámbito de una unidad de trabajo? ¿Puede una sola Session de Hibernate extenderse a través de varias transacciones de la base de datos o ésta es una relación uno-a-uno de ámbitos? ¿Cuándo debe abrir y cerrar una Session? y ¿cómo demarca los límites de la transacción de la base de datos?

- **Unidad de trabajo**

Primero, no use el antipatrón *sesión-por-operación*: no abra y cierre una Session para cada llamada simple a la base de datos en un solo hilo. Lo mismo aplica para las transacciones de base de datos. Las llamadas a la base de datos en una aplicación se hacen usando una secuencia planeada; estas se agrupan dentro de unidades de trabajo atómicas. Esto también significa que el auto-commit después de cada una de las declaraciones SQL es inútil en una aplicación ya que este modo está pensado para trabajo ad-hoc de consola SQL. Hibernate deshabilita, o espera que el servidor de aplicaciones lo haga, el modo auto-commit inmediatamente. Las transacciones de las bases de datos nunca son opcionales. Toda comunicación con una base de datos tiene que ocurrir dentro de una transacción. El comportamiento auto-commit para leer datos se debe evitar, ya que hay muy poca probabilidad de que las transacciones pequeñas funcionen mejor que una unidad de trabajo definida claramente. La última es mucho más sostenible y extensible.

El patrón más común en una aplicación multiusuario cliente/servidor es *sesión-por-petición*. En este modelo, una petición del cliente se envia al servidor, en donde se ejecuta la capa de persistencia de Hibernate. Se abre una nueva Session de Hibernate y todas las operaciones de la base de datos se ejecutan en esta unidad de trabajo. Una vez completado el trabajo, y una vez se ha preparado la respuesta para el cliente, se limpia la sesión y se cierra. Use una sóla transacción de la base de datos para servir la petición del cliente, dándole inicio y guardándola cuando abre y cierra la Session. La relación entre las dos es uno-a-uno y este modelo es a la medida perfecta de muchas aplicaciones.

El reto se encuentra en la implementación. Hibernate brinda administración incorporada de la "sesión actual" para simplificar este patrón. Inicie una transacción cuando se tiene que procesar un pedido del servidor y termine la transacción antes de

que se envie la respuesta al cliente. Las soluciones más comunes son ServletFilter, un interceptor AOP con un punto de corte en los métodos del servicio o un contenedor proxy/intercepción. Un contenedor EJB es una manera estandarizada de implementar aspectos de doble filo como demarcación de transacción en beans de sesión EJB, declarativamente con CMT.

Puede extender el ámbito de una Session y transacción de la base de datos hasta que "se ha presentado la vista". Esto es bastante útil en aplicaciones de servlet que utilizan una fase de entrega separada después de que se ha procesado el pedido. El extender la transacción de la base de datos hasta que la entrega de la vista se encuentre completa es fácil de lograr si implementa su propio interceptor. Sin embargo, no se logra fácilmente si depende de EJBs con transacciones administradas por el contenedor. Una transacción se completará cuando un método EJB retorna, antes de que pueda empezar la entrega de cualquier vista. Vea el sitio web de Hibernate y el foro para encontrar consejos y ejemplos sobre este patrón de *sesión abierta en vista*.

3.16.2 CONVERSACIONES LARGAS

El patrón sesión-por-peticiónno es la única forma de diseñar unidades de trabajo. Muchos procesos empresariales requieren una serie completa de interacciones con el usuario intercaladas con accesos a la base de datos. En aplicaciones empresariales y web no es aceptable que una transacción de la base de datos abarque la interacción de un usuario. Considere el siguiente ejemplo:

- Se abre la primera pantalla de un diálogo. Los datos que ve el usuario han sido cargados en una Session en particular y en una transacción de la base de datos. El usuario es libre de modificar los objetos.
- El usuario hace click en "Guardar" después de 5 minutos y espera que sus modificaciones se hagan persistentes. También espera que él sea la única

persona editando esta información y que no ocurra ningún conflicto en la modificación.

Desde el punto de vista del usuario, llamamos a esta unidad de trabajo, una larga conversación o transacción de aplicación. Hay muchas formas de implementar esto en su aplicación.

Una primera implementación ingenua podría mantener abierta la Session y la transacción de la base de datos durante el tiempo para pensar del usuario, con bloqueos en la base de datos para prevenir la modificación simultánea y para garantizar el aislamiento y la atomicidad. Esto es un antipatrón, ya que la contención de bloqueo no permitiría a la aplicación escalar con el número de usuarios simultáneos.

Tiene que usar varias transacciones de la base de datos para implementar la conversación. En este caso, mantener el aislamiento de los procesos empresariales se vuelve una responsabilidad parcial de la capa de la aplicación. Una sóla conversación usualmente abarca varias transacciones de la base de datos. Será atómica si sólo una de estas transacciones de la base de datos (la última) almacena los datos actualizados. Todas las otras simplemente leen datos (por ejemplo, en un diálogo de estilo-asistente abarcando muchos ciclos petición/respuesta). Esto es más fácil de implementar de lo que suena, especialmente si usa las funcionalidades de Hibernate:

- **Versionado automático**: Hibernate puede realizar un control automático de concurrencia optimista por usted.Puede detectar automáticamente si ha ocurrido una modificación simultánea durante el tiempo para pensar del usuario.
- **Objetos separados:** Si decide usar el patrón *sesión-por-petición*, todas las instancias cargadas estarán en estado separado durante el tiempo para pensar del usuario. Hibernate le permite volver a unir los objetos y hacer persistentes las modificaciones. El patrón se llama *sesión-por-petición-con-objetos-*

separados. Se usa el versionado automático para aislar las modificaciones simultáneas.

- **Sesión extendida (o larga)**:La Session de Hibernate puede ser desconectada de la conexión JDBC subyacente después de que haya guardado la transacción de la base de datos y haya reconectado cuando ocurra una nueva petición del cliente. Este patrón se conoce como *sesión-por-conversación* y hace la re-unión innecesaria. Para aislar las modificaciones simultáneas se usa el versionado automático y usualmente no se permite que se limpie la Session automáticamente sino explícitamente.

Tanto la *sesión-por-petición-con-objetos-separados* como la *sesión-por-conversación* tienen ventajas y desventajas.

3.17. MANEJO DE EXCEPCIONES

Si la Session lanza una excepción, incluyendo cualquier SQLException, debe deshacer inmediatamente la transacción de la base de datos, llamar a Session.close() y descartar la instancia de Session. Ciertos métodos de Session *no* dejarán la sesión en un estado consistente. Ninguna excepción lanzada por Hibernate puede ser tratada como recuperable. Asegúrese de que la Session se cierre llamando a close() en un bloque finally.

La HibernateException, que envuelve a la mayoría de los errores que pueden ocurrir en la capa de persistencia de Hibernate, es una excepción no chequeada. No lo era en versiones anteriores de Hibernate. En nuestra opinión, no debemos forzar al desarrollador de aplicaciones a capturar una excepción irrecuperable en una capa baja. En la mayoría de los sistemas, las excepciones no chequeadas y fatales son manejadas en uno de los primeros cuadros de la pila de llamadas a métodos (por ejemplo, en las capas más altas) y presenta un mensaje de error al usuario de la

aplicación o se toma alguna otra acción apropiada. Note que Hibernate podría también lanzar otras excepciones no chequeadas que no sean una HibernateException. Estas no son recuperables y debe tomarse una acción apropiada.

Hibernate envuelve SQLExceptions lanzadas mientras se interactúa con la base de datos en una JDBCException. De hecho, Hibernate intentará convertir la excepción en una subclase de JDBCException más significativa. La SQLException subyacente siempre está disponible por medio de JDBCException.getCause(). Hibernate convierte la SQLException en una subclase de JDBCException apropiada usando el SQLExceptionConverter adjunto a la SessionFactory. Por defecto, el SQLExceptionConverter está definido por el dialecto configurado. Sin embargo, también es posible enchufar una implementación personalizada. Los subtipos estándar de JDBCException son:

- JDBCConnectionException: indica un error con la comunicación JDBC subyacente.
- SQLGrammarException: indica un problema de gramática o sintáxis con el SQL publicado.
- ConstraintViolationException: indica alguna forma de violación de restricción de integridad.
- LockAcquisitionException: indica un error adquiriendo un nivel de bloqueo necesario para realizar una operación solicitada.
- GenericJDBCException: una excepción genérica que no encajó en ninguna de las otras categorías.

3.18. TIEMPO DE ESPERA DE LA TRANSACCIÓN

Una característica importante proporcionada por un entorno administrado como EJB que nunca es proporcionado para un código no-administrado, es el tiempo de espera de la transacción. Estos tiempos de espera se aseguran de que ninguna transacción

que se comporte inapropiadamente pueda vincular recursos mientras no devuelva una respuesta al usuario. Fuera de un entorno administrado (JTA), Hibernate no puede proporcionar completamente esta funcionalidad. Sin embargo, Hibernate puede por lo menos controlar las operaciones de acceso de datos, asegurándose de que los bloqueos a nivel de base de datos y las consultas con grandes grupos de resultados se encuentran limitados por un tiempo de espera definido. En un entorno administrado, Hibernate puede delegar el tiempo de espera de la transacción a JTA. Esta funcionalidad es abstraída por el objeto Transaction de Hibernate.

```
Session sess = factory.openSession();
try {

    //set transaction timeout to 3 seconds
    sess.getTransaction().setTimeout(3);
    sess.getTransaction().begin();
    // do some work
    ...
    sess.getTransaction().commit()
}
catch (RuntimeException e) {
    sess.getTransaction().rollback();
    throw e; // or display error message
}
finally {
    sess.close();
}
```

setTimeout()

No se puede llamar en un bean CMT, en donde se deben definir declarativamente los tiempos de espera de las transacciones.

3.19. CONTROL DE CONCURRENCIA OPTIMISTA

El único enfoque consistente con una alta concurrencia y una alta escalabilidad es el control de concurrencia optimista con versionamiento. El chequeo de versión utiliza números de versión, o sellos de fecha (timestamps), para detectar actualizaciones en conflicto y para prevenir la pérdida de actualizaciones. Hibernate proporciona tres enfoques posibles de escribir código de aplicación que utilice concurrencia optimista. Los casos de uso que mostramos se encuentran en el contexto de conversaciones largas, pero el chequeo de versiones tiene además el beneficio de prevenir la pérdida de actualizaciones en transacciones individuales de la base de datos.

3.20. MODOS DE LIBERACIÓN DE LA CONEXIÓN

La herencia (2x) de Hibernate en relación con la administración de la conexion JDBC fue que una Session obtendría una conexión cuando se necesitara por primera vez y luego la mantendría hasta que se cerrara la sesión. Hibernate 3.x introdujo la noción de modos de liberación de conexión para decirle a la sesión como manejar sus conexiones JDBC. La siguiente discusión sólamente es pertinente para las conexiones provistas por medio de un ConnectionProvider configurado. Los diferentes modos de liberación se identifican por los valores numerados de org.Hibernate.ConnectionReleaseMode:

- ON_CLOSE: es el comportamiento heredado descrito anteriormente. La sesión de Hibernate obtiene una conexión cuando necesita acceder a JDBC la primera vez y mantiene esa conexión hasta que se cierra la sesión.
- AFTER_TRANSACTION: libera las conecciones después de que se ha completado una org.Hibernate.Transaction.

- AFTER_STATEMENT (también se conoce como una liberación agresiva): libera conexiones después de cada ejecución de una declaración. Se salta esta liberación agresiva si la declaración deja abiertos recursos asociados con la sesión dada. Actualmente la única situación donde ocurre esto es por medio del uso de org.Hibernate.ScrollableResults.

El parámetro de configuración Hibernate.connection.release_mode se utiliza para especificar el modo de liberación a utilizar. Los valores posibles son los siguientes:

- auto (predeterminado): esta opción delega al modo de liberación devuelto por el métodoorg.Hibernate.transaction.TransactionFactory.getDefaultReleaseMode().
- Para JTATransactionFactory, esto devuelve ConnectionReleaseMode.AFTER_STATEMENT; para JDBCTransactionFactory, esto devuelve ConnectionReleaseMode.AFTER_TRANSACTION. No cambie este comportamiento predeterminado ya que las fallas debido a este valor de esta configuración tienden a indicar errores y/o suposiciones en el código del usuario.
- on_close: usa ConnectionReleaseMode.ON_CLOSE. Esta configuración se deja para la compatibilidad con versiones anteriores, pero no se recomienda para nada su utilización.
- after_transaction: utiliza ConnectionReleaseMode.AFTER_TRANSACTION. Esta configuración no se debe utilizar en entornos JTA. También note que con ConnectionReleaseMode.AFTER_TRANSACTION, si se considera que una sesión se encuentra en modo auto-commit, las conexiones serán liberada como si el modo de liberación fuese AFTER_STATEMENT.
- after_statement: usa ConnectionReleaseMode.AFTER_STATEMENT. Además se consulta la ConnectionProvider configurada para ver si soporta esta característica supportsAggressiveRelease(). Si no, el modo de liberación se

vuelve a establecer como ConnectionReleaseMode.AFTER_TRANSACTION. Esta configuración sólamente es segura en entornos en donde podemos re-adquirir la misma conexión JDBC subyacente cada vez que llamamos a ConnectionProvider.getConnection() o en entornos auto-commit, en donde no importa si recibimos la misma conexión.

3.21. TIPOS DE CONSULTAS QUE SOPORTA HIBERNATE

3.21.1. HQL

Hibernate utiliza un lenguaje de consulta potente (HQL) que se parece a SQL. Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación.

- **Sensibilidad a mayúsculas**

Las consultas no son sensibles a mayúsculas, a excepción de los nombres de las clases y propiedades Java. De modo que SeLeCT es lo mismo que sELEct e igual a SELECT, pero org.Hibernate.eg.FOO no es lo mismo que org.Hibernate.eg.Foo y foo.barSet no es igual a foo.BARSET.

Este manual utiliza palabras clave HQL en minúsculas. Algunos usuarios encuentran que las consultas con palabras clave en mayúsculas son más fáciles de leer, pero esta convención no es apropiada para las peticiones incluidas en código Java.

- **La cláusula from**

La consulta posible más simple de Hibernate es de esta manera:

```
from eg.Cat
```

Esto retorna todas las instancias de la clase eg.Cat. Usualmente no es necesario calificar el nombre de la clase ya que auto-import es el valor predeterminado. Por ejemplo:

```
from Cat
```

Con el fin de referirse al Cat en otras partes de la petición, necesitará asignar un *alias*. Por ejemplo:

```
from Cat as cat
```

Esta consulta asigna el alias cat a las instancias Cat, de modo que puede utilizar ese alias luego en la consulta. La palabra clave as es opcional. También podría escribir:

```
from Cat cat
```

Pueden aparecer múltiples clases, lo que causa un producto cartesiano o una unión "cruzada" (cross join).

```
from Formula, Parameter
from Formula as form, Parameter as param
```

Se considera como una buena práctica el nombrar los alias de consulta utilizando una inicial en minúsculas, consistente con los estándares de nombrado de Java para las variables locales (por ejemplo, domesticCat).

- **Asociaciones y uniones (joins)**

También puede asignar alias a entidades asociadas o a elementos de una colección de valores utilizando una join. Por ejemplo:

```
from Cat as cat
   inner join cat.mate as mate
   left outer join cat.kittens as kitten
from Cat as cat left join cat.mate.kittens as kittens
from Formula form full join form.parameter param
```

Los tipos de uniones soportadas se tomaron prestados de ANSI SQL

- inner join
- left outer join
- right outer join
- full join (no es útil usualmente)

Las construcciones inner join, left outer join y right outer join se pueden abreviar.

```
from Cat as cat
    join cat.mate as mate
    left join cat.kittens as kitten
```

Puede proveer condiciones extras de unión utilizando la palabra clave with de HQL.

```
from Cat as cat
   left join cat.kittens as kitten
      with kitten.bodyWeight > 10.0
from Cat as cat
   inner join fetch cat.mate
left join fetch cat.kittens
```

Usualmente no se necesita asignársele un alias a una unión de recuperación ya que los objetos asociados no se deben utilizar en la cláusula where (ni en cualquier otra cláusula). Los objetos asociados no se retornan directamente en los resultados de la consulta. En cambio, se pueden acceder por medio del objeto padre. La única razón por la que necesitaríamos un alias es si estamos uniendo recursivamente otra colección:

```
from Cat as cat
   inner join fetch cat.mate
   left join fetch cat.kittens child
left join fetch child.kittens
```

La construcción fetch no puede utilizarse en consultas llamadas que usen iterate() (aunque se puede utilizar scroll()). Fetch se debe usar junto con setMaxResults() o setFirstResult() ya que estas operaciones se basan en las filas de resultados, las

cuales usualmente contienen duplicados para la recuperación de colección temprana, por lo tanto, el número de filas no es lo que se esperaría. Fetch no se debe usar junto con una condición with improvisadas. Es posible crear un producto cartesiano por medio de una recuperación por union más de una colección en una consulta, así que tenga cuidado en este caso. La recuperación por unión de múltiples roles de colección también da resultados a veces inesperados para mapeos de bag, así que tenga cuidado de cómo formular sus consultas en este caso. Finalmente, observe que full join fetch y right join fetch no son significativos.

Si está utilizando una recuperación perezosa a nivel de propiedad (con instrumentación de código byte), es posible forzar a Hibernate a traer las propiedades perezosas inmediatamente utilizando fetch all properties.

```
from Document fetch all properties order by name
from Document doc fetch all properties where lower(doc.name) like '%cats%'
```

- **Formas de sintaxis unida**

HQL soporta dos formas de unión de asociación: implicit y explicit.

Las consultas que se mostraron en la sección anterior todas utilizan la forma explicit, en donde la palabra clave join se utiliza explícitamente en la claúsula from. Esta es la forma recomendada.

La forma implicit no utiliza la palabra clave join. Las asociaciones se "desreferencian" utilizando la notación punto. Uniones implicit pueden aparecer en cualquiera de las cláusulas HQL. La unión implicit causa uniones internas (inner joins) en la declaración SQL que resulta.

```
from Cat as cat where cat.mate.name like '%s%'
```

- **Referencia a la propiedad identificadora**

Hay dos maneras de referirse a la propiedad identificadora de una entidad:

- La propiedad especial (en minúsculas) id se puede utilizar para referenciar la propiedad identificadora de una entidad dado que la entidad no defina un id del nombre de la propiedad no-identificadora.
- Si la entidad define una propiedad identificadora nombrada, puede utilizar ese nombre de propiedad.

Las referencias a propiedades identificadoras compuestas siguen las mismas reglas de nombramiento. Si la entidad no tiene un id del nombre de la propiedad no-identificadora, la propiedad identificadora compuesta sólamente puede ser referenciada por su nombre definido. De otra manera se puede utilizar la propiedad id especial para referenciar la propiedad identificadora.

- **La cláusula select**

La cláusula select escoge qué objetos y propiedades devolver en el conjunto de resultados de la consulta. Considere lo siguiente:

```
select mate from Cat as cat  inner join cat.mate as mate
```

La consulta seleccionará mates de otros Cats. Puede expresar esta consulta de una manera más compacta así:

```
select cat.mate from Cat cat
```

Las consultas pueden retornar propiedades de cualquier tipo de valor incluyendo propiedades del tipo componente:

```
select cat.name from DomesticCat cat
where cat.name like 'fri%'
select cust.name.firstName from Customer as cust
```

Las consultas pueden retornar múltiples objetos y/o propiedades como un array de tipo Object[].

```
select mother, offspr, mate.name
from DomesticCat as mother
inner join mother.mate as mate
left outer join mother.kittens as offspr
```

O como un List:

```
select new list(mother, offspr, mate.name)
from DomesticCat as mother
inner join mother.mate as mate
left outer join mother.kittens as offspr
```

O asumiendo que la clase Family tiene un constructor apropiado - como un objeto Java de tipo seguro:

```
select new Family(mother, mate, offspr)
from DomesticCat as mother
 join mother.mate as mate
 left join mother.kittens as offspr
```

Puede asignar alias para expresiones seleccionadas utilizando as:

```
select max(bodyWeight) as max, min(bodyWeight) as min,
count(*) as n
from Cat cat
```

Esto es lo más útil cuando se usa junto con select new map:

```
select new map( max(bodyWeight) as max,
min(bodyWeight) as min, count(*) as n ) from Cat cat
```

Esta consulta devuelve un Map de alias a valores seleccionados.

- **Funciones de agregación**

Las consultas HQL pueden incluso retornar resultados de funciones de agregación sobre propiedades:

```
select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat)
from Cat cat
```

Las funciones de agregación soportadas son:

- avg(...), sum(...), min(...), max(...)
- count(*)
- count(...), count(distinct ...), count(all...)

Puede utilizar operadores aritméticos, concatenación y funciones SQL reconocidas en la cláusula select:

```
select cat.weight + sum(kitten.weight)
from Cat cat
join cat.kittens kitten
group by cat.id, cat.weight
select firstName||' '||initial||' '||upper(lastName) from Person
```

Las palabras clave distinct y all se pueden utilizar y tienen las mismas semánticas que en SQL.

```
select distinct cat.name from Cat cat

select count(distinct cat.name), count(cat) from Cat cat
```

- **La cláusula where**

La cláusula where le permite refinar la lista de instancias retornadas. Si no existe ningún alias, puede referirse a las propiedades por nombre:

```
from Cat where name='Fritz'
```

Si existe un alias, use un nombre de propiedad calificado:

```
from Cat as cat where cat.name='Fritz'
```

Esto retorna instancias de Cat llamadas 'Fritz'.

La siguiente petición:

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

Retornará todas las instancias de Foo con una instancia de bar con una propiedad date igual a la propiedad startDate del Foo. Las expresiones de ruta compuestas hacen la cláusula where extremadamente potente. Tome en consideración lo siguiente:

```
from Cat cat where cat.mate.name is not null
```

Esta consulta se traduce a una consulta SQL con una unión de tabla (interna). Por ejemplo:

```
from Foo foo
where foo.bar.baz.customer.address.city is not null
```

La segunda consulta es eficiente y no se necesita una unión de tablas.

También se pueden utilizar las propiedades de identificadores compuestos. Considere el siguiente ejemplo en donde Person tiene identificadores compuestos que consisten de country y medicareNumber:

```
from bank.Person person where person.id.country = 'AU'
and   person.id.medicareNumber = 123456
```

Una vez más, la segunda consulta no requiere una unión de tablas.

La propiedad especial class acccede al valor discriminador de una instancia en el caso de persistencia polimórfica. Un nombre de clase Java incluído en la cláusula where será traducido a su valor discriminador.

```
from Cat cat where cat.class = DomesticCat
```

Un tipo "any" tiene las propiedades especiales id y class, permiténdole expresar una unión de la siguiente forma (en donde AuditLog.item es una propiedad mapeada con <any>).

```
from AuditLog log, Payment payment
where log.item.class = 'Payment' and log.item.id = payment.id
```

La log.item.class y payment.class harían referencia a los valores de columnas de la base de datos completamente diferentes en la consulta anterior.

```
from DomesticCat cat where cat.name between 'A' and 'B'
from DomesticCat cat where cat.name in ( 'Foo', 'Bar', 'Baz' )
```

Las formas negadas se pueden escribir así:

```
from DomesticCat cat where cat.name not between 'A' and 'B'
from DomesticCat cat where cat.name not in ( 'Foo', 'Bar', 'Baz' )
```

De manera similar, is null y is not null se pueden utilizar para probar valores nulos.

Los valores booleanos se pueden utilizar fácilmente en expresiones declarando substituciones de consulta HQL en la configuración de Hibernate:

```
<property name="Hibernate.query.substitutions"
>true 1, false 0</property>
```

Esto remplazará las palabras clave true y false con los literales 1 y 0 en el SQL traducido de este HQL:

```
from Cat cat where cat.alive = true
```

Puede comprobar el tamaño de una colección con la propiedad especial size o la función especial size().

```
from Cat cat where cat.kittens.size> 0
from Cat cat where size(cat.kittens) > 0
```

Para las colecciones indexadas, puede referirse a los índices máximo y mínimo utilizando las funciones minindex y maxindex. De manera similar, se puede referir a los elementos máximo y mínimo de una colección de tipo básico utilizando las funciones minelement y maxelement. Por ejemplo:

```
from Calendar cal where maxelement(cal.holidays)> current_date
from Order order where maxindex(order.items)> 100
from Order order where minelement(order.items)> 10000
```

Las funciones SQL any, some, all, exists, in están soportadas cuando se les pasa el conjunto de elementos o índices de una colección (las funciones elements e indices) o el resultado de una subconsulta (vea a continuación):

```
select mother from Cat as mother, Cat as kit
where kit in elements(foo.kittens)
select p from NameList list, Person p
where p.name = some elements(list.names)
from Cat cat where exists elements(cat.kittens)
```

Note que estas construcciones - size, elements, indices, minindex, maxindex, minelement, maxelement - solo se pueden utilizar en la cláusula where en Hibernate3. Los elementos de colecciones indexadas (arrays, listas, mapas) se pueden referir por índice sólamente en una cláusula where:

```
from Order order where order.items[0].id = 1234
select person from Person person, Calendar calendar
where calendar.holidays['national day'] = person.birthDay
   and person.nationality.calendar = calendar
select item from Item item, Order order
where order.items[ order.deliveredItemIndices[0] ] = item and order.id = 11
select item from Item item, Order order
where order.items[ maxindex(order.items) ] = item and order.id = 11
```

La expresión dentro de [] puede incluso ser una expresión aritmética:

```
select item from Item item, Order order
where order.items[ size(order.items) - 1 ] = item
where order.items[ maxindex(order.items) ] = item and order.id = 11
```

HQL también proporciona la función incorporada index(), para los elementos de una asociación uno-a-muchos o una colección de valores.

```
select item, index(item) from Order order
   join order.items item
where index(item) < 5
```

Se pueden utilizar las funciones SQL escalares soportadas por la base de datos subyacente:

```
from DomesticCat cat where upper(cat.name) like 'FRI%'
```

Considere qué tan larga y menos leíble sería la siguiente consulta en SQL:

```
select cust from Product prod,    Store store     inner join store.customers cust
where prod.name = 'widget'     and store.location.name in ( 'Melbourne', 'Sydney' )
and prod = all elements(cust.currentOrder.lineItems)
SELECT cust.name, cust.address, cust.phone, cust.id, cust.current_order
FROM customers cust,  stores store,    locations loc,   store_customers sc,  product prod
WHERE prod.name = 'widget'     AND store.loc_id = loc.id AND loc.name IN ( 'Melbourne',
'Sydney' )    AND sc.store_id = store.id   AND sc.cust_id = cust.id AND prod.id = ALL(SELECT
item.prod_id  FROM line_items item, orders o
 WHERE item.order_id = o.id  AND cust.current_order = o.id)
```

- **Cláusula order by**

La lista retornada por una consulta se puede ordenar por cualquier propiedad de una clase retornada o componentes:

```
from DomesticCat cat
order by cat.name asc, cat.weight desc, cat.birthdate
```

Los asc o desc opcionales indican ordenamiento ascendente o descendente respectivamente.

- **Cláusula group by**

Una consulta que retorna valores agregados se puede agrupar por cualquier propiedad de una clase retornada o componentes:

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat
group by cat.color
select foo.id, avg(name), max(name)
from Foo foo join foo.names name
group by foo.id
```

Se permite también una cláusula having.

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat
group by cat.color
having cat.color in (eg.Color.TABBY, eg.Color.BLACK)
```

Las funciones SQL y las funciones de agregación SQL están permitidas en las cláusulas having y order by, si están soportadas por la base de datos subyacente (por ejemplo, no lo están en MySQL).

```
select cat
from Cat cat
   join cat.kittens kitten
group by cat.id, cat.name, cat.other, cat.properties
having avg(kitten.weight)
> 100
order by count(kitten) asc, sum(kitten.weight) desc
```

La cláusula group by ni la cláusula order by pueden contener expresiones aritméticas. Hibernate tampocoo expande una entidad agrupada así que no puede escribir group by cat si todas las propiedades de cat son no-agregadas. Tiene que enumerar todas las propiedades no-agregadas explícitamente.

- **Subconsultas**

Para bases de datos que soportan subconsultas, Hibernate soporta subconsultas dentro de consultas. Una subconsulta se debe encerrar entre paréntesis (frecuentemente por una llamada a una función de agregación SQL). Incluso se permiten subconsultas correlacionadas (subconsultas que se refieren a un alias en la consulta exterior).

```
from Cat as fatcat
where fatcat.weight
> (
   select avg(cat.weight) from DomesticCat cat
)
from DomesticCat as cat
where cat.name = some (
   select name.nickName from Name as name
)
from Cat as cat
where not exists (
   from Cat as mate where mate.mate = cat
)
from DomesticCat as cat
where cat.name not in (
   select name.nickName from Name as name
)
```

- **Componentes**

Los componentes se pueden utilizar de la misma manera en que se pueden utilizar los tipos de valores simples en consultas HQL. Pueden aparecer en la cláusula select así:

```
select p.name from Person p
select p.name.first from Person p
```

En donde el nombre de la Persona es un componente. Los componentes también se pueden utilizar en la cláusula where:

```
from Person p where p.name = :name
from Person p where p.name.first = :firstName
```

Los componentes también se pueden utilizar en la cláusula where:

```
from Person p order by p.name
from Person p order by p.name.first
```

Otro uso común de los componentes se encuentra en row value constructors.

- **Sintaxis del constructor de valores por fila**

HQL soporta la utilización de la sintaxis row value constructor de SQL ANSI que a veces se denomina sintaxis tuple, aunque puede que la base de datos subyacentes no soporte esa noción. Aquí estamos refiriéndonos generalmente a las comparaciones multivaluadas que se asocian típicamente con los componentes. Considere una entidad Persona, la cual define un componente de nombre:

```
from Person p where p.name.first='John' and p.name.last='Jingleheimer-Schmidt'
```

Esa es una sintaxis válida aunque un poco verbosa. Puede hacerlo un poco más conciso utilizando la sintaxis row value constructor:

```
from Person p where p.name=('John', 'Jingleheimer-Schmidt')
```

También puede ser útil especificar esto en la cláusula select:

```
select p.name from Person p
```

También puede ser beneficioso el utilizar la sintaxis row value constructor cuando se utilizan subconsultas que necesitan compararse con valores múltiples:

```
from Cat as cat
where not ( cat.name, cat.color ) in (select cat.name, cat.color from DomesticCat cat)
```

Algo que se debe tomar en consideración al decidir si quiere usar esta sintaxis es que la consulta dependerá del orden de las sub-propiedades componentes en los metadatos.

También puede expresar sus consultas en el dialecto SQL nativo de su base de datos. Esto es útil si quiere utilizar las características especificas de la base de datos tales como hints de consulta o la palabra clave CONNECT en Oracle. También proporciona una ruta de migración limpia desde una aplicación basada en SQL/JDBC a Hibernate.

Hibernate3 le permite especificar SQL escrito a mano, incluyendo procedimientos almacenados para todas las operaciones create, update, delete y load.

3.21.2 SQL

La ejecución de consultas SQL nativas se controla por medio de la interfaz SQLQuery, la cual se obtiene llamando a Session.createSQLQuery(). Las siguientes secciones describen cómo utilizar esta API para consultas.

- **Consultas Escalares**

```
sess.createSQLQuery("SELECT * FROM CATS").list();
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").list();
```

La consulta SQL más básica es para obtener a una lista de escalares (valores).

Estas retornarán una lista de objetos arrays (Object[]) con valores escalares para cada columna en la tabla CATS. Hibernate utilizará ResultSetMetadata para deducir el orden real y los tipos de los valores escalares retornados.

Para evitar los gastos generales de la utilización de ResultSetMetadata o simplemente para ser más explícito en lo que se devuelve se puede utilizar addScalar():

```
sess.createSQLQuery("SELECT * FROM CATS")
 .addScalar("ID", Hibernate.LONG)
 .addScalar("NAME", Hibernate.STRING)
 .addScalar("BIRTHDATE", Hibernate.DATE)
```

Se especifica esta consulta:

- la cadena de consulta SQL
- las columnas y tipos que se devuelven

Esto retornará objetos arrays, pero no utilizará ResultSetMetdata sino que obtendrá explícitamente las columnas de IDENTIFICACION, NOMBRE y FECHA DE NACIMIENTO respectivamente como Larga, Cadena y Corta del grupo de resultados subyacente. Esto también significa que sólamente estas tres columnas serán retornadass aunque la consulta este utilizando * y pueda devolver más de las tres columnas enumeradas.

Es posible dejar afuera la información de tipo para todos o algunos de los escalares.

```
sess.createSQLQuery("SELECT * FROM CATS")
 .addScalar("ID", Hibernate.LONG)
 .addScalar("NAME")
 .addScalar("BIRTHDATE")
```

Esto es esencialmente la misma consulta que antes, pero ahora se utiliza ResultSetMetaData para determinar el tipo de NOMBRE y FECHA DE NACIMIENTO, mientras que el tipo de IDENTIFICACION se especifica explícitamente.

El dialecto controla la manera en que los java.sql.Types retornados de ResultSetMetaData se mapean a los tipos de Hibernate. Si un tipo en especial no se encuentra mapeado o no resulta en el tipo esperado es posible personalizarlo por medio de llamadas a registerHibernateType en el dialecto.

- **Consultas de entidades**

Todas las consultas anteriores eran sobre los valores escalraes devueltos, basicamente devolviendo los valores "crudos" desde el grupo resultado. Lo siguiente muestra como obtener los objetos entidades desde una consulta sql nativa por medio de addEntity().

```
sess.createSQLQuery("SELECT * FROM CATS").addEntity(Cat.class);
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").addEntity(Cat.class);
```

Se especifica esta consulta:

- la cadena de consulta SQL
- la entidad devuelta por la consulta

Asumiendo que Cat es mapeado como una clase con las columnas IDENTIFICACION, NOMBRE y FECHA DE NACIMIENTO las consultas anteriores devolverán una Lista en donde cada elemento es una entidad Cat.

Si la entidad es mapeada con una many-to-one a otra entidad tambien se necesita que devuelva esto cuando realice una consulta nativa, de otra manera, aparecerá un error "no se encontró la columna" específico a la base de datos. Se devolverán automáticamente las columnas adicionales cuando se utiliza la anotación *, pero preferimos ser tan explícitos así como lo muestra el siguiente ejemplo para una many-to-one a un Dog:

Esto permitirá que cat.getDog() funcione apropiadamente.

```
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE, DOG_ID FROM CATS").addEntity(C
at.class);
```

- **Manejo de asociaciones y colecciones**

Es posible unir de manera temprana en el Dog para evitar el posible viaje de ida y vuelta para iniciar el proxy. Esto se hace por medio del método addJoin(), el cual le permite unirse en una asociación o colección.

```
sess.createSQLQuery("SELECT c.ID, NAME, BIRTHDATE, DOG_ID, D_ID, D_NAME FROM CATS c,
 DOGS d WHERE c.DOG_ID = d.D_ID")
 .addEntity("cat", Cat.class)
 .addJoin("cat.dog");
```

En este ejemplo los Cats retornados tendrán su propiedad dog completamente iniciada sin ningún viaje extra de ida y vuelta a la base de datos. Observe que agregó un nombre alias ("cat") para poder especificar la ruta de la propiedad de destino de la unión. Es posible hacer la misma unión temprana para colecciones, por ejemplo, si el Cat tuviese en lugar un Dog uno-a-muchos.

```
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE, D_ID, D_NAME, CA
T_ID FROM CATS c, DOGS d WHERE c.ID = d.CAT_ID")
 .addEntity("cat", Cat.class)
 .addJoin("cat.dogs");
```

En este punto estamos alcanzando los límites de lo que es posible con las consultas nativas sin empezar a mejorar las consultas sql para hacerlas utilizables en Hibernate. Los problemas empiezan a surgir cuando las entidades múltiples retornadas son del mismo tipo o cuando no son suficientes los nombres de las columnas/alias predeterminados.

- **Devolución de entidades múltiples**

Hasta ahora se ha asumido que los nombres de las columnas del grupo de resultados son las mismas que los nombres de columnas especificados en el documento de mapeo. Esto puede llegar a ser problemático para las consultas SQL que unen

múltiples tablas ya que los mismos nombres de columnas pueden aparecer en más de una tabla.

Se necesita una inyección de alias en las columnas en la siguiente consulta (que con mucha probabilidad fallará):

```
sess.createSQLQuery("SELECT c.*, m.*  FROM CATS c, CATS m
WHERE c.MOTHER_ID = c.ID") .addEntity("cat", Cat.class)
 .addEntity("mother", Cat.class)
```

La intención de esta consulta es retornar dos instancias Cat por fila: un gato y su mamá. Sin embargo, esto fallará debido a que hay un conflicto de nombres;las instancias se encuentran mapeadas a los mismos nombres de columna. También en algunas bases de datos los alias de las columnas retornadas serán con mucha probabilidad de la forma "c.IDENTIFICACION", "c.NOMBRE", etc, los cuales no son iguales a las columnas especificadas en los mapeos ("IDENTIFICACION" y "NOMBRE").

La siguiente forma no es vulnerable a la duplicación de nombres de columnas:

```
sess.createSQLQuery("SELECT {cat.*}, {mother.*}  FROM CATS c, CATS m
WHERE c.MOTHER_ID = c.ID")
.addEntity("cat", Cat.class)
.addEntity("mother", Cat.class)
```

Se especifica esta consulta:

- la cadena de consultas SQL, con un espacio reservado para que Hibernate inserte alias de columnas
- las entidades devueltas por la consulta

La anotación {cat.*} y {mother.*} que se utilizó anteriormente es la abreviatura para "todas las propiedades". Opcionalmente puede enumerar las columnas explícitamente, pero inclusive en este caso Hibernate inyecta los alias de columnas SQL para cada propiedad. El espacio para un alias de columna es sólamente el nombre calificado de la propiedad del alias de la tabla. En el siguiente ejemplo, recuperamos Cats y sus madres desde una tabla diferente (cat_log) a la declarada en los meta datos de mapeo. Inclusive puede utilizar los alias de propiedad en la cláusula where.

```
String sql = "SELECT ID as {c.id}, NAME as {c.name}, " +
        "BIRTHDATE as {c.birthDate}, MOTHER_ID as {c.mother}, {mother.*} " +
        "FROM CAT_LOG c, CAT_LOG m WHERE {c.mother} = c.ID";

List loggedCats = sess.createSQLQuery(sql)
       .addEntity("cat", Cat.class)
       .addEntity("mother", Cat.class).list()
```

- **Referencias de propiedad y alias**

Para la mayoría de los casos, se necesita la inyección del alias anterior. Para las consultas relacionadas con mapeos más complejos como propiedades compuestas, discriminadores de herencia, colecciones, etc, existen alias especificos a utilizar para permitir que Hibernate inyecte los alias apropiados.

La siguiente tabla muestra las diferentes maneras de utilizar la inyección de alias. Note que los nombres alias en el resultado son simplemente ejemplos; cada alias tendrá un nombre único y probablemente diferente cuando se utilice.

Tabla III.IV. Nombres con inyección alias

Descripción	Sintaxis	Ejemplo
Una propiedad	{[aliasname].[propertyname]	A_NAME as {item.name}

Descripción	**Sintaxis**	**Ejemplo**
simple		
Una propiedad compuesta	{[aliasname].[componentname].[propertyname]}	CURRENCY as {item.amount.currency}, VALUE as {item.amount.value}
Discriminador de una entidad	{[aliasname].class}	DISC as {item.class}
Todas las propiedades de una entidad	{[aliasname].*}	{item.*}
Una clave de colección	{[aliasname].key}	ORGID as {coll.key}
La identificación -id- de una colección	{[aliasname].id}	EMPID as {coll.id}
El elemento de una colección	{[aliasname].element}	XID as {coll.element}
propiedad del elemento	{[aliasname].element.[propertyname]}	NAME as {coll.element.name}

Descripción	Sintaxis	Ejemplo
en la colección		
Todas las propiedades del elemeto en la colección	{[aliasname].element.*}	{coll.element.*}
Todas las propiedades de la colección	{[aliasname].*}	{coll.*}

Fuente:http://docs.jboss.org/Hibernate/core/3.6/reference/es-ES/html_single/

- **Devolución de entidades no-administradas**

Es posible aplicar un ResultTransformer para consultas SQL nativas, permitiéndole retornar entidades no-administradas.

```
sess.createSQLQuery("SELECT NAME, BIRTHDATE FROM CATS")
.setResultTransformer(Transformers.aliasToBean(CatDTO.class))
```

Se especifica esta consulta:

- la cadena de consulta SQL
- un transformador de resultado

La consulta anterior devolverá una lista de CatDTO a la cual se ha instanciado e inyectado los valores de NOMBRE y FECHA DE NACIMIENTO en su propiedades o campos correspondientes.

- **Manejo de herencias**

Las consultas SQL nativas, las cuales consultan por entidades que son mapeadas como parte de una herencia tienen que incluir todas las propiedades para la clase base y todas sus subclases.

- **Parámetros**

Las consultas SQL nativas soportan parámetros nombrados así como posicionales:

```
Query query = sess.createSQLQuery("SELECT * FROM CATS WHERE NAME like ?").addEntity(Cat.cl
ass);
List pusList = query.setString(0, "Pus%").list();

query = sess.createSQLQuery("SELECT * FROM CATS WHERE NAME like :name").addEntity(Cat.cla
ss);
List pusList = query.setString("name", "Pus%").list();
```

- **Consultas SQL nombrada**

Las consultas SQL nombradas se pueden definir en el documento de mapeo y se pueden llamar de la misma manera que una consulta HQL nombrada. En este caso, *no* necesitamos llamar a addEntity ().

```
<sql-query name="persons">
   <return alias="person" class="eg.Person"/>
   SELECT person.NAME AS {person.name},
         person.AGE AS {person.age},
         person.SEX AS {person.sex}
   FROM PERSON person
   WHERE person.NAME LIKE :namePattern
</sql-query
>
List people = sess.getNamedQuery("persons")
   .setString("namePattern", namePattern)
   .setMaxResults(50)
   .list();
```

El elemento <return-join> se utiliza para unir asociaciones y el elemento <load-collection> se usa para definir consultas, las cuales dan inicio a colecciones.

```
<sql-query name="personsWith">
   <return alias="person" class="eg.Person"/>
   <return-join alias="address" property="person.mailingAddress"/>
   SELECT person.NAME AS {person.name},
         person.AGE AS {person.age},
         person.SEX AS {person.sex},
         address.STREET AS {address.street},
         address.CITY AS {address.city},
         address.STATE AS {address.state},
         address.ZIP AS {address.zip}
   FROM PERSON person
   JOIN ADDRESS address
      ON person.ID = address.PERSON_ID AND address.TYPE='MAILING'
   WHERE person.NAME LIKE :namePattern
</sql-query>
```

Una consulta SQL nombrada puede devolver un valor escalar. Tiene que declarar el alias de la columna y el tipo de Hibernate utilizando el elemento <return-scalar>:

```
<sql-query name="mySqlQuery">
   <return-scalar column="name" type="string"/>
   <return-scalar column="age" type="long"/>
   SELECT p.NAME AS name,
         p.AGE AS age,
   FROM PERSON p WHERE p.NAME LIKE 'Hiber%'
</sql-query
>
```

Puede externalizar el grupo de resultados mapeando información en un elemento <resultset>, el cual le permitirá reutilizarlos a través de consultas nombradas o por medio de la API setResultSetMapping().

```
<resultset name="personAddress">
   <return alias="person" class="eg.Person"/>
   <return-join alias="address" property="person.mailingAddress"/>
</resultset>
<sql-query name="personsWith" resultset-ref="personAddress">
   SELECT person.NAME AS {person.name},
         person.AGE AS {person.age},
         person.SEX AS {person.sex},
         address.STREET AS {address.street},
         address.CITY AS {address.city},
         address.STATE AS {address.state},
         address.ZIP AS {address.zip}
   FROM PERSON person
   JOIN ADDRESS address
      ON person.ID = address.PERSON_ID AND address.TYPE='MAILING'
   WHERE person.NAME LIKE :namePattern
</sql-query>
```

- **Utilización de procedimientos para consultas**

Hibernate 3 brinda soporte para consultas por medio de procedimientos almacenados y funciones. La mayoría de la siguiente documentación es igual para ambos. La función/procedimiento almacenado tiene que retornar un grupo de resultados como el primer parámetro de salida para poder trabajar con Hibernate. A continuación hay un ejemplo de tal función almacenada en Oracle 9 y posteriores:

```
CREATE OR REPLACE FUNCTION selectAllEmployments
   RETURN SYS_REFCURSORAS    st_cursor SYS_REFCURSOR;
BEGIN
   OPEN st_cursor FOR
 SELECT EMPLOYEE, EMPLOYER,
 STARTDATE, ENDDATE,
 REGIONCODE, EID, VALUE, CURRENCY
 FROM EMPLOYMENT;
    RETURN  st_cursor;
 END;
```

Para utilizar esta consulta en Hibernate se necesita mapearla por medio de una consulta nombrada.

```
<sql-query name="selectAllEmployees_SP" callable="true">
   <return alias="emp" class="Employment">
      <return-property name="employee" column="EMPLOYEE"/>
      <return-property name="employer" column="EMPLOYER"/>
      <return-property name="startDate" column="STARTDATE"/>
      <return-property name="endDate" column="ENDDATE"/>
      <return-property name="regionCode" column="REGIONCODE"/>
      <return-property name="id" column="EID"/>
      <return-property name="salary">
         <return-column name="VALUE"/>
         <return-column name="CURRENCY"/>
      </return-property>
   </return>
   { ? = call selectAllEmployments() }
</sql-query>
```

Los procedimientos almacenados actualmente sólo retornan escalares y entidades. No se soporta <return-join> ni <load-collection>.

- **Personalice SQL para cargar**

También puede declararsuspropiasSQL (o HQL)para la carga deconsultasentidad.Al igual que conlas inserciones, actualizacionesy eliminaciones,esto se puede hacera nivel decolumna individualo en elnivel de los estados. He aquí unejemplo de unadeclaraciónde anulaciónnivel:

```
<sql-query name="person">
   <return alias="pers" class="Person" lock-
mode="upgrade"/>
   SELECT NAME AS {pers.name}, ID AS {pers.id}
   FROM PERSON
   WHERE ID=?
   FOR UPDATE
</sql-query>
```

Esta es tan sólo una declaración de consulta nombrada, como se discutió anteriormente. Puede referenciar esta consulta nombrada en un mapeo de clase:

```
<class name="Person">
    <id name="id">
        <generator class="increment"/>
    </id>
    <property name="name" not-null="true"/>
    <loader query-ref="person"/>
</class>
```

Esto funciona inclusive con procedimientos almacenados.

Puede incluso definir una consulta para la carga de colección:

```
<set name="employments" inverse="true">
    <key/>
    <one-to-many class="Employment"/>
    <loader query-ref="employments"/>
</set>
<sql-query name="employments">
    <load-
collection alias="emp" role="Person.employments"/>
    SELECT {emp.*}
```

También puede definir un cargador de entidad que cargue una colección con una unión temprana:

```
<sql-query name="person">
    <return alias="pers" class="Person"/>
    <return-
join alias="emp" property="pers.employments"/>
    SELECT NAME AS {pers.*}, {emp.*}
    FROM PERSON pers
    LEFT OUTER JOIN EMPLOYMENT emp
        ON pers.ID = emp.PERSON_ID
    WHERE ID=?
</sql-query>
```

3.21.3 QBC

Acompaña a Hibernate una API de consultas por criterios intuitiva y extensible.

- **Creación de una instancia Criteria**

La interfaz org.Hibernate.Criteria representa una consulta contra una clase persistente en particular. La Session es una fábrica de instancias de Criteria.

```
Criteria crit = sess.createCriteria(Cat.class);
crit.setMaxResults(50);
List cats = crit.list();
```

Límitando el conjunto de resultados

Un criterio individual de consulta es una instancia de la interfaz org.Hibernate.criterion.Criterion. La clase org.Hibernate.criterion.Restrictions define métodos de fábrica para obtener ciertos tipos incorporados de Criterion.

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .add( Restrictions.between("weight", minWeight, maxWeight) )
    .list();
```

Las restricciones se pueden agrupar lógicamente.

```
List cats = sess.createCriteria(Cat.class)
   .add( Restrictions.like("name", "Fritz%") )
   .add( Restrictions.or(
      Restrictions.eq( "age", new Integer(0) ),
      Restrictions.isNull("age")
   ) )
   .list();
List cats = sess.createCriteria(Cat.class)
```

```
    .add( Restrictions.in( "name", new String[] { "Fritz", "Izi", "Pk" } ) )
    .add( Restrictions.disjunction()
        .add( Restrictions.isNull("age") )
        .add( Restrictions.eq("age", new Integer(0) ) )
        .add( Restrictions.eq("age", new Integer(1) ) )
        .add( Restrictions.eq("age", new Integer(2) ) )
    ) )
    .list();
```

Hay un rango de tipos de criterios incorporados (subclases de Restrictions). Uno de los más útiles le permite especificar SQL directamente.

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.sqlRestriction("lower({alias}.name) like lower(?)", "Fritz%",
Hibernate.STRING) )    .list();
```

El sitio {alias} será remplazado por el alias de fila de la entidad consultada.

También puede obtener un criterio de una instancia Property. Puede crear una Property llamando a Property.forName().

```
Property age = Property.forName("age");
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.disjunction()
        .add( age.isNull() )
        .add( age.eq( new Integer(0) ) )
        .add( age.eq( new Integer(1) ) )
        .add( age.eq( new Integer(2) ) )
    ) )
    .add( Property.forName("name").in( new String[] {
"Fritz", "Izi", "Pk" } ) )
    .list();
```

- **Orden de los resultados**

Puede ordenar los resultados usando org.Hibernate.criterion.Order.

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
    .setMaxResults(50)
    .list();
List cats = sess.createCriteria(Cat.class)
    .add( Property.forName("name").like("F%") )
    .addOrder( Property.forName("name").asc() )
    .addOrder( Property.forName("age").desc() )
    .setMaxResults(50)
    .list();
```

- **Asociaciones**

Al navegar asociaciones usando createCriteria() puede especificar restricciones en entidades relacionadas:

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%") )
    .createCriteria("kittens")
        .add( Restrictions.like("name", "F%") )
    .list();
```

El segundo createCriteria() retorna una nueva instancia de Criteria, que se refiere a los elementos de la colección kittens.

Hay una alternativa que es útil en ciertas circunstancias:

```
List cats = sess.createCriteria(Cat.class)
    .createAlias("kittens", "kt")
    .createAlias("mate", "mt")
    .add( Restrictions.eqProperty("kt.name", "mt.name") )
    .list();
```

(createAlias() no crea una nueva instancia de Criteria.)

Las colecciones de gatitos de las instancias Cat retornadas por las dos consultas previas *no* están prefiltradas por los criterios. Si desea recuperar sólo los gatitos que coincidan con los criterios debe usar un ResultTransformer.

```
List cats = sess.createCriteria(Cat.class)
    .createCriteria("kittens", "kt")
        .add( Restrictions.eq("name", "F%") )
    .setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP)
    .list();
Iterator iter = cats.iterator();
while ( iter.hasNext() ) {
    Map map = (Map) iter.next();
    Cat cat = (Cat) map.get(Criteria.ROOT_ALIAS);
    Cat kitten = (Cat) map.get("kt");
}
```

Adicionalmente puede manipular el grupo de resultados utilizando una unión externa izquierda:

```
List cats = session.createCriteria( Cat.class )
                .createAlias("mate",        "mt",        Criteria.LEFT_JOIN,
Restrictions.like("mt.name", "good%") )
                .addOrder(Order.asc("mt.age"))
.list();
```

Esto retornará todos los Cats -gatos- con una pareja cuyo nombre empiece por "good" ordenado de acuerdo a la edad de la pareja y todos los cats -gatos- que no tengan una pareja. Esto es útil cuando hay necesidad de ordenar o limitar en la base de datos antes de retornar grupos de resultados complejos/grandes y elimina muchas instancias en donde se tendrían que realizar múltiples consultas y unir en memoria los resultados por medio de java.

Sin esta funcionalidad, primero todos los cats sin una pareja tendrían que cargarse en una petición.

Una segunda petición tendría que recuperar los cats -gatos- con las parejas cuyos nombres empiecen por "good" ordenado de acuerdo a la edad de las parejas.

Tercero, en memoria sería necesario unir las listas manualmente.

- **Proyecciones, agregación y agrupamiento**

La clase org.Hibernate.criterion.Projections es una fábrica de instancias de Projection. Puede aplicar una proyección a una consulta llamando a setProjection().

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.rowCount() )
    .add( Restrictions.eq("color", Color.BLACK) )
    .list();
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.projectionList()
        .add( Projections.rowCount() )
        .add( Projections.avg("weight") )
        .add( Projections.max("weight") )
        .add( Projections.groupProperty("color") )
    )
    .list();
```

No es necesario ningún "agrupamiento por" explícito en una consulta por criterios. Ciertos tipos de proyecciones son definidos para ser *proyecciones agrupadas*, que además aparecen en la cláusula SQL group by.

Puede asignar un alias a una proyección de modo que el valor proyectado pueda ser referido en restricciones u ordenamientos. Aquí hay dos formas diferentes de hacer esto:

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.alias( Projections.groupProperty("color"), "colr" ) )
    .addOrder( Order.asc("colr") )
    .list();
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.groupProperty("color").as("colr") )
    .addOrder( Order.asc("colr") )
    .list();
```

Los métodos alias() y as() simplemente envuelven una instancia de proyección en otra instancia de Projection con alias. Como atajo, puede asignar un alias cuando agregue la proyección a una lista de proyecciones:

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.projectionList()
        .add( Projections.rowCount(), "catCountByColor" )
        .add( Projections.avg("weight"), "avgWeight" )
        .add( Projections.max("weight"), "maxWeight" )
        .add( Projections.groupProperty("color"), "color" )
    )
    .addOrder( Order.desc("catCountByColor") )
    .addOrder( Order.desc("avgWeight") )
    .list();
List results = session.createCriteria(Domestic.class, "cat")
    .createAlias("kittens", "kit")
    .setProjection( Projections.projectionList()
        .add( Projections.property("cat.name"), "catName" )
        .add( Projections.property("kit.name"), "kitName" )
    )
    .addOrder( Order.asc("catName") )
    .addOrder( Order.asc("kitName") )
    .list();
```

También puede usar Property.forName() para expresar proyecciones:

```
List results = session.createCriteria(Cat.class)
    .setProjection( Property.forName("name") )
    .add( Property.forName("color").eq(Color.BLACK) )
    .list();
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.projectionList()
        .add( Projections.rowCount().as("catCountByColor") )
        .add( Property.forName("weight").avg().as("avgWeight") )
        .add( Property.forName("weight").max().as("maxWeight") )
        .add( Property.forName("color").group().as("color" )
    )
    .addOrder( Order.desc("catCountByColor") )
    .addOrder( Order.desc("avgWeight") )
    .list();
```

- **Consultas y subconsultas separadas**

La clase DetachedCriteria le permite crear una consulta fuera del ámbito de una sesión y luego ejecutarla usando una Session arbitraria.

```
DetachedCriteria query = DetachedCriteria.forClass(Cat.class)
   .add( Property.forName("sex").eq('F') );

Session session = ....;
Transaction txn = session.beginTransaction();
List results = query.getExecutableCriteria(session).setMaxResults(100).list();
txn.commit();
session.close();
```

También puede utilizar una DetachedCriteria para expresar una subconsulta. Las instancias de Criterio involucrando en subconsultas se pueden obtener por medio de Subqueries o Property.

```
DetachedCriteria avgWeight = DetachedCriteria.forClass(Cat.class)
    .setProjection( Property.forName("weight").avg() );
session.createCriteria(Cat.class)
    .add( Property.forName("weight").gt(avgWeight) )
    .list();
DetachedCriteria weights = DetachedCriteria.forClass(Cat.class)
    .setProjection( Property.forName("weight") );
session.createCriteria(Cat.class)
    .add( Subqueries.geAll("weight", weights) )
    .list();
```

Las subconsultas correlacionadas tambień son posibles:

```
DetachedCriteria avgWeightForSex = DetachedCriteria.forClass(Cat.class, "cat2")
    .setProjection( Property.forName("weight").avg() )
    .add( Property.forName("cat2.sex").eqProperty("cat.sex") );
session.createCriteria(Cat.class, "cat")
    .add( Property.forName("weight").gt(avgWeightForSex) )
    .list();
```

- **Consultas por identificador natural**

Para la mayoría de las consultas, incluyendo las consultas por criterios, el caché de consulta no es muy eficiente debido a que la invalidación del caché de consulta ocurre con demasiada frecuencia. Sin embargo, hay un tipo especial de consulta donde podemos optimizar el algoritmo de invalidación de caché: búsquedas de una clave natural constante. En algunas aplicaciones, este tipo de consulta, ocurre frecuentemente. La API de criterios brinda una provisión especial para este caso.

Primero, mapee la clave natural de su entidad utilizando <natural-id> y habilite el uso del caché de segundo nivel.

```
<class name="User">
    <cache usage="read-write"/>
    <id name="id">
        <generator class="increment"/>
    </id>
    <natural-id>
        <property name="name"/>
        <property name="org"/>
    </natural-id>
    <property name="password"/>
</class>
```

Esta funcionalidad no está pensada para uso con entidades con claves naturales *mutables*.

Una vez que haya habilitado el caché de consulta de Hibernate, Restrictions.naturalId() le permite hacer uso del algoritmo de caché más eficiente.

```
session.createCriteria(User.class)
    .add( Restrictions.naturalId()
        .set("name", "gavin")
        .set("org", "hb")
    ).setCacheable(true)
    .uniqueResult();
```

3.22.4 QBE

La búsqueda mediante ejemplo (QBE-Query by Example) es una lengua de consulta de bases de datos relacionados similar al lenguaje de consulta estructurado (SQL). Este sistema nos permite que la persona que genera la búsqueda o la aplicación que la realice pueda proporcionar información sobre aquello que está buscando en la base de datos. QBE fue ideado por Moshé M.Zloof en IBM Research durante la década de los 70 de manera conjunta con SQL, siendo el primer idioma gráfico de consulta. Originalmente sólo podía realizar recuperaciones de datos aunque posteriormente se amplió a otros tipos de conceptos como pueden ser imágenes.

- **Funcionamiento**

El sistema QBE consiste en que el usuario introduce, mediante un formulario, información para realizar la búsqueda utilizando tablas de ejemplo. QBE usa variables de dominio como en DRC para crear estas tablas. De esta manera conseguimos que el usuario forme parte de la consulta, ya que el origen de la búsqueda se encuentra en la mente de éste. Esto ayuda a poder minimizar los errores de la búsqueda ya que es el propio usuario que nos facilita información sobre aquello que tiene en mente. A continuación mostramos una tabla de ejemplo:

Tabla III.V. Ejemplo de QBE

Empresa	Responsable	Dirección	Teléfono
A	XXX	RRRRRRRR	22222222
B	YYY	TTTTTTTT	33333333
C	ZZZ	PPPPPPPP	44444444
D	SSS	QQQQQQQQ	88888888

Fuente: http://pt.wikipedia.org/wiki/Query_by_Example

Mediante una búsqueda QBE y utilizando como ejemplo la tabla anterior, el usuario podría proporcionar la variable empresa y recibir toda la información adicional que se desee saber cómo puede ser el contacto, el teléfono o la dirección.

- **Descriptores**

Es importante saber qué tipos de descriptores se pueden utilizar para la realización de cualquier búsqueda.

- **Nivel bajo:** Son aquellos descriptores que utilizan información exclusiva del contenido del ejemplo proporcionado. Un ejemplo de este tipo de descriptores sería el caso de las imágenes. De éstas podríamos obtener información mediante el histograma, colores, textura…
- **Nivel medio:** Usan información de contexto del objeto buscado. Utilizando el ejemplo anterior de la imagen, este tipo de descriptores nos proporcionaría información como la fecha de creación, la web donde se encontró…
- **Nivel alto:** Son aquellos descriptores que utilizan metadatos. En este nivel encontramos las etiquetas manuales que se anotan en la imagen como puede ser el nombre del autor.

- **Tipos de búsqueda con ejemplo**

Como hemos mencionado anteriormente hay diferentes tipos de descriptores, por lo que podemos realizar variaos tipos de búsqueda. Actualmente QBE se utiliza para la búsqueda de la mayoría de conceptos debido a su simplicidad de utilización para el usurario y su correcta presentación de resultados. Podemos utilizar QBE en búsquedas como por ejemplo:

- **Búsqueda con texto**

Este es el tipo de búsqueda más antiguo. Consiste en dar al sistema cualquier información y que éste devuelva entradas relacionas con el tema. Como hemos comentado, el buscador con texto es el más antiguo ya que fueron los primeros buscadores que se pudieron utilizar. Estos consisten en proporcionar al sistema cualquier tipo de texto y éste retornar información de cualquier tipo ya sean imágenes, videos. El sistema de búsqueda más conocido es el PageRank que utiliza "Google".

- **Búsqueda con imágenes**

Es una técnica de consulta que implica dotar al sistema CBIR de un ejemplo como texto, imagen. El sistema CBIR realiza la comparación del contenido del ejemplo con los que tenemos en la base de datos mediante las tablas de comparación mencionadas anteriormente. Un ejemplo de esta aplicación podría ser la búsqueda de imágenes mediante una sola imagen. El CBIR utiliza datos como el histograma, el color, la textura, los metadatos asociados a la imagen o información añadida por el usuario ofreciendo resultados de similares características. Una aplicación actual que proporciona este servicio podría ser “Google Imágenes”. Este tipo de sistema también es utilizado en búsqueda de imágenes para móviles.

- **Búsqueda de vídeo**

Son sistemas informáticos diseñados para buscar videos guardados en dispositivos digitales con podría ser servidores o bases de datos. Las búsquedas se realizan

utilizando la indexación audiovisual. Estos tipos de sistemas utilizan varios tipos de criterios de búsqueda dependiendo de la naturaleza del buscador o del objetivo de éste. Los tipos de criterios son:

- **Metadatos:** informaciones sobre datos concretos como puede ser el título, el autor etc.
- **Reconocimiento de voz:** Consiste en una transcripción del audio del vídeo para que una vez analizados, el sistema es capaz de saber la temática de la secuencia y de proporcionar la información al usuario.
- **Reconocimiento de texto:** Se utilizan este tipo de criterios para reconocer personajes de los videos mediante de chyrons.
- **Análisis de fotogramas:** Como el sistema de video es una sucesión de imágenes se puede extraer información de estos utilizando los sistemas de búsqueda de imágenes.

- **Búsqueda con audio**

Estos sistemas consisten en que el usuario introduce cualquier tipo de información para realizar una búsqueda de audio. La información podría ser un fichero de audio, texto, imagen.Retornando el sistema audios relacionados o similares cumpliendo los requisitos iniciales. Dentro de este tipo de buscadores podemos encontrar los de audio mediante un ejemplo de audio. Estas aplicaciones estudian el audio que aporta el usuario, extrayendo todo tipo de información como la frecuencia, el tono, las voces, los datos asociado y devuelve un listado de canciones, previamente introducidas en la base de datos, que cumplen las condiciones que el internauta solicita. Un ejemplo de este tipo de aplicación seria el "Shazam". Este programa es un buscador de títulos de canciones. El usuario proporciona al sistema un audio, éste lo analiza y retorna el título de la canción.

- **Búsqueda multimodal**

Es un sistema muy parecido a la búsqueda por ejemplo ya que también hay una interacción entre el usuario y el sistema buscador. La búsqueda multimodal consiste en una interficie que permite enviar consultas de búsqueda no sólo mediante peticiones de texto sino también mediante otros medios como imágenes, vídeos… además de tener en cuenta otros aspectos como el contexto dónde se encuentra el usuario.

3.22 VENTAJAS Y DESVENTAJAS

Ventajas

- Productividad: Evita mucho del código confuso de la capa de persistencia, permitiendo centrarse en la lógica de negocio.
- Mantenibilidad: Por tener pocas líneas de código permite que el código sea más claro. Al dividir la capa de persistencia se puede identificar los errores muy fácilmente.
- Rendimiento: Existe la tendencia a pensar que una solución “manual” es más eficiente que una “automática”. Hibernate tiene un buen desempeño pero todo depende realmente de cómo se realicen las consultas y como se configure el Framework.
- Indecencia del proveedor: Una solución ORM te abstrae del SGBD. Permite desarrollar en local con bases de datos ligeras sin implicación en el entorno de producción.

Desventajas

- Hibernate sólo impone una condición para poder usarse, las tablas deben tener una clave primaria, preferentemente una clave no natural, pero debe poder identificar los registros de alguna manera.

- Si bien Hibernate reúne un conjunto de características que lo hacen ser muy llamativo, no es una solución óptima en el caso de proyectos de migración de datos, y si los criterios con que se crearon la base de datos no tienen un mínimo de calidad.

CAPÍTULO IV
4. ANALISIS COMPARATIVO

4.1 INTRODUCCION

Los sistemas de computación están cada vez más integrados a las necesidades de la empresa moderna, ya no sólo para modelar procesos manuales sino también aprovechando la inmensa capacidad de análisis de información disponible. A medida que la complejidad de estos sistemas crece, es crucial que los componentes que modelan la lógica sean aislados de las distintas tecnologías utilizadas en la solución.

En sus comienzos, la Ingeniería de Software atacó esta problemática separando la naturaleza de los datos de sus procesos asociados. Una herencia de este principio son las bases de datos relacionales, verdaderos repositorios donde se mantienen esquemas modelando los datos y un lenguaje propio para manipularlos.

La programación orientada a objetos rompe con esta separación y fuerza el igual tratamiento de los datos y los procedimientos, basándose en los principios de encapsulación y ocultamiento de la información. Estos lenguajes proveen facilidades muy primitivas para que los objetos sobrevivan a una ejecución del programa, característica esencial de una aplicación empresarial.

En la actualidad la industria del software se enfrenta con los problemas inherentes causados por la integración de estas dos tecnologías: los modelos orientados a objetos y las bases de datos relacionales, distribuyendo así las responsabilidades de modelar la lógica y persistir los objetos. Esta integración impone verdaderos retos al momento de la construcción, ya que si bien comparten algunas características, existen disparidades en la representación y manejo de la información que imponen limitaciones e introducen costos de desarrollo y mantenimiento.

El presente capítulo tiene como objetivo el estudio de la persistencia de objetos Java en una base de datos relacional, utilizando para ello una de las soluciones disponibles para esta integración denominada ORM (Mapeo Objeto – Relacional), en donde la persistencia se realiza de forma totalmente automática y transparente a la lógica de la aplicación.

4.2 DETERMINACIÓN DE LAS HERRAMIENTAS A COMPARAR

En la actualidad existen muchos frameworks de persistencia para el desarrollo de aplicaciones web usando el lenguaje de programación JAVA.

Los Frameworks seleccionados para el análisis son Hibernate y EclipseLink, a continuación ofrecemos una descripción rápida de las mismas.

4.2.1 HIBERNATE

Hibernate es una herramienta ORM completa que ha conseguido en un tiempo record una excelente reputación en la comunidad de desarrollo posicionándose claramente como el producto OpenSource3 líder en este campo gracias a sus prestaciones, buena documentación y estabilidad. Es valorado por muchos incluso como solución superior a productos comerciales dentro de su enfoque, siendo una muestra clara de su reputación y soporte la reciente integración dentro del grupo JBOSS4 que

seguramente generará iniciativas muy interesantes para el uso de Hibernate dentro de este servidor de aplicaciones.

Hibernate parte de una filosofía de mapear objetos Java "normales", también conocidos en la comunidad como "POJO's" (Plain Old Java Objects). No contempla la posibilidad de automatizar directamente la persistencia de Entity Beans tipo BMP (es decir, generar automáticamente este tipo de objetos), aunque aún así es posible combinar Hibernate con este tipo de beans utilizando los patrones para la delegación de persistencia en POJO's.

Una característica de la filosofía de diseño de Hibernate ha de ser destacada especialmente, dada su gran importancia: puede utilizar los objetos Java definidos por el usuario tal cual, es decir, no utiliza técnicas como generación de código a partir de descriptores del modelo de datos o manipulación de bytecodes en tiempo de compilación (técnica conocida por su amplio uso en JDO), ni obliga a implementar interfaces determinados, ni heredar de una superclase. Utiliza en vez de ello el mecanismo de reflexión de Java. Hibernate trata con la reflexión de Java y el aumento de clases en tiempo de ejecución utilizando una librería de generación de código Java muy poderosa y de alto rendimiento llamada CGLIB. CGLIB se utiliza para extender clases Java e implementar interfaces Java en tiempo de ejecución.

4.2.2 ECLIPSELINK

EclipseLink es la fuente abierta de Eclipse Persistencia Proyecto de Servicios de la Fundación Eclipse . El software proporciona un marco extensible que permite a los desarrolladores de Java para interactuar con diversos servicios de datos, incluyendo bases de datos, servicios web, objeto de asignación de XML (OXM), y sistemas de información empresarial (EIS). EclipseLink es compatible con varios estándares de persistencia, incluyendo:

- Java Persistence API (JPA)
- Arquitectura Java para XML Binding (JAXB)
- Java Connector Architecture (JCA)
- Service Data Objects (SDO).

EclipseLink se basa en la Eclipselink producto de que Oracle ha contribuido al código fuente para crear el proyecto EclipseLink. La contribución original de la base de código 11g Eclipselink, y todo el conjunto de code-base/feature fue aportado, con sólo EJB Container 2 -Managed Persistence (CMP) y algunos menores de Oracle Application Server elimina integración específica. Esto difiere de la Eclipselink Essentials GlassFish contribución, que no incluía algunas de las características clave de la empresa. Los nombres de los paquetes se han cambiado y una parte del código y la configuración se ha movido alrededor.

4.3. ANÁLISIS DE LAS HERRAMIENTAS

4.3.1. HIBERNATE

Introducción

En Java los datos se representan en objetos. Sin embargo, las bases de datos habituales (como Oracle) guardan sus datos en forma relacional. Evidentemente existe una brecha entre estos dos mundos ("objetos-relacional") que, de alguna manera, debe completarse.

Los Frameworks que se encargan de adaptar el mundo de objetos al relacionan son conocidos como ORM (Object-Relational Mapping). Existen varios ORM en el mercado, si bien Hibernate es el ORM que se convirtió en un estandard-de-facto.

Hibernate se encarga, justamente, de relacionar clases con tablas. A forma muy simple, una tabla se mapea contra una clase, y cada columna contra un atributo de dicha clase.

De esta forma, Hibernate se encargará de ocultar la complejidad del acceso a datos, exponiendo solamente objetos. Idealmente, en una aplicación con Hibernate, no es necesario generar querys SQL para interactuar con los datos (de hecho, la aplicación no tiene contacto directo con la base de datos).

Hibernate es un entorno de trabajo que tiene como objetivo facilitar la persistencia de objetos Java en bases de datos relacionales y al mismo tiempo la consulta de estas bases de datos para obtener objetos.

Hibernate es un Framework que agiliza la relación entre la aplicación y la base de datos. De todos los Frameworks ORM que se ha utilizado, sin dudas es el más completo.

Características Principales

- No intrusivo (estilo POJO)
- Soluciona el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional).
- Es flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente.
- Muy buena documentación (fórums para ayuda, libro)
- Comunidad activa con muchos usuarios
- Transacciones, caché, asociaciones, polimorfismo, herencia, lazy loading, persistencia transitiva, estrategias de fetching.
- Potente lenguaje de consulta (HQL): subqueries, outer joins, ordering, proyeccion (report query), paginacion.
- Facil testeo.

- No es standard.
- Persistencia de Java Beans

Arquitectura

El API de Hibernate es una arquitectura de dos capas (Capa de persistencia y Capa de Negocio).

El diagrama a continuación brinda una perspectiva a alto nivel de la arquitectura de Hibernate:

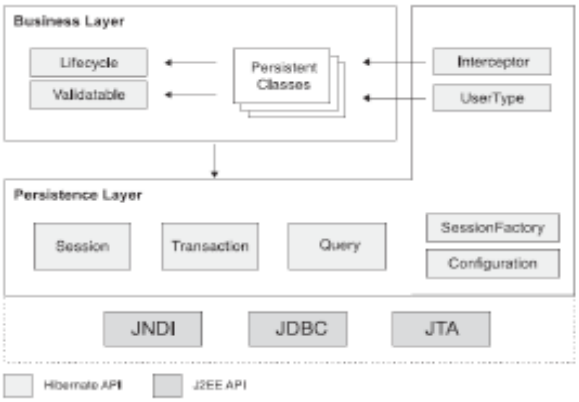


Figura IV.14.Arquitectura de Hibernate

Fuente: http://docs.jboss.org/Hibernate/core/3.6/reference/es-ES/html_single/

Hibernate es flexible y soporta diferentes enfoques. Sin embargo, mostraremos los dos extremos: la arquitectura "mínima" y la arquitectura "completa".

Este diagrama ilustra la manera en que Hibernate utiliza la base de datos y los datos de configuración para proporcionar servicios de persistencia y objetos persistentes a la aplicación.

Componentes de Hibernate

- **Connection Management (Gestión de conexión).**

Hibernate proporciona un servicio de gestión de la conexión para proporcionar una eficiente conexión a la base de dato.

La Conexión de base de datos es la parte más cara de interactuarya que requiere una gran cantidad de recursos para abrir ycerrar la conexión.

- **Transaction management: (Gestión de transacciones)**

Proporciona la capacidad al usuario ejecutar más de una base de datos a la vez.

- **Object relational mapping:** (Mapeo objeto relacional)

Es una técnica de programación que representa los datos deun modelo de datos relacional de forma de objeto.Esta parte de la utiliza para seleccionar, insertar, actualizary borrar los registros de la tabla subyacente. Cuando pasamos de un objeto a un Session.save () método, Hibernate lee elestado de las variables de ese objeto y ejecuta la consultanecesaria.

- **Interfaces**

Las interfaces mostradas pueden clasificarse como sigue:

Interfaces llamadas por la aplicación:

Para realizar operaciones básicas (inserciones, borrados, consultas): Session, Transaction, y Query.

Interfaces llamadas por el código de la infraestructura de la aplicación para configurar Hibernate.

La más importante es la clase Configuration.

Interfaces callback:

Que permiten a la aplicación reaccionar ante determinados eventos que ocurren dentro de la aplicación, tales como Interceptor, Lifecycle, y Validatable.

Interfaces que permiten extender las funcionalidades de mapeado de Hibernate.

Como por ejemplo UserType, CompositeUserType, e IdentifierGenerator.

Además, Hibernate hace uso de APIs de Java, tales como JDBC, JTA (Java Transaction Api) y JNDI (Java Naming Directory Interface).

Interfaz Session

Es una de las interfaces primarias en cualquier aplicación Hibernate.

Una instancia de Session es "poco pesada" y su creación y destrucción es muy "barata".

Esto es importante, ya que nuestra aplicación necesitará crear y destruir sesiones todo el tiempo, quizá en cada petición. Puede ser útil pensar en una sesión como en una caché o colección de objetos cargados (a o desde una base de datos) relacionados con una única unidad de trabajo. Hibernate puede detectar cambios en los objetos pertenecientes a una unidad de trabajo.

Interfaz SessionFactory

Permite obtener instancias Session. Esta interfaz no es "ligera", y debería compartirse entre muchos hilos de ejecución. Típicamente hay una única SessionFactory para toda la aplicación, creada durante la inicialización de la misma. Sin embargo, si la aplicación accede a varias bases de datos se necesitará una SessionFactory por cada base de datos.

La interfaz Configuration

Se utiliza para configurar y "arrancar" Hibernate. La aplicación utiliza una instancia de Configuration para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de Hibernate, y a continuación crea la SessionFactory.

La interfaz Query

Permite realizar peticiones a la base de datos y controla cómo se ejecuta dicha petición (query). Las peticiones se escriben en HQL o en el dialecto SQL nativo de la base de datos que estemos utilizando. Una instancia Query se utiliza para enlazar los parámetros de la petición, limitar el número de resultados devueltos por la petición, y para ejecutar dicha petición.

Type

Un objeto Type Hibernate hace corresponder un tipo Java con un tipo de unacolumna de la base de datos. Todas las propiedades persistentes de las clases persistentes, incluyendo las asociaciones, tienen un tipo Hibernate correspondiente. Este diseño haceque Hibernate sea altamente flexible y extendible. Incluso se permiten tipos definidos porel usuario (interfaz UserType y CompositeUserType).

Ventajas

- Productividad: Evita mucho del código confuso de la capa de persistencia, permitiendo centrarse en la lógica de negocio.
- Mantenibilidad: Por tener pocas líneas de código permite que el código sea más claro. Al dividir la capa de persistencia se puede identificar los errores muy fácilmente.
- Rendimiento: Existe la tendencia a pensar que una solución “manual” es más eficiente que una “automática”. Hibernate tiene un buen desempeño pero todo depende realmente de cómo se realicen las consultas y como se configure el Framework.
- Hibernate puede trabajar como proveedor de JPA en aplicaciones basadas en la APP.
- Independiente del DBS se puede utilizar cualquier base de datos a elección.
- Creación de SQL automático: Crea el SQL automáticamente y deja libre al desarrollador de mapear el resultado del query hacia el objeto.
- Hibernate soporta la herencia, asociaciones y colecciones.
- Hibernate tiene la capacidad de generar claves primarias automáticamente mientras se almacena registros en la base de datos.
- Hibernate tiene su propio lenguaje de consulta independiente de la base de datos.
- Hibernate soporta anotaciones aparte de XML.

- La obtención de la paginación en Hibernate es bastante simple.
- Collections: Puede guardar y recuperar colecciones enteras (Map, Set, ArrayList) o un objeto entero que en sus atributos tenga colecciones.
- Sistema de Cache de objetos: Así se puede reutilizar objetos que ya recuperaste y se ahorra muchas consultas a la base de datos.
- Se puede integrar con otros framework de desarrollo de aplicaciones web.
- Hibernate cuenta con un manejador de excepciones lo cual permite detectar errores y solucionarlos de manera fácil y rápida.

Desventajas

- Hibernate sólo impone una condición para poder usarse, las tablas deben tener una clave primaria, preferentemente una clave no natural, pero debe poder identificar los registros de alguna manera.
- Si bien Hibernate reúne un conjunto de características que lo hacen ser muy llamativo, no es una solución óptima en el caso de proyectos de migración de datos, y si los criterios con que se crearon la base de datos no tienen un mínimo de calidad.

4.3.2 ECLIPSELINK

Introducción

EclipseLink es una implementación de código abierto de Java Persistence API (JPA) especificación que incluye extensiones más allá de lo que se define en la APP. EclipseLink también incluye la API de especificación Java para XML Binding (JAXB). Estas extensiones incluyen propiedades de persistencia de la unidad, las sugerencias de consulta, anotaciones y API personalizado. Las extensiones de metadatos XML EclipseLink están contenidas en Moxy, un componente EclipseLink que permite enlazar las clases Java a esquemas XML.

Características Principales

- No intrusivo, flexible, basada en metadatos arquitectura
- Apoyo a la cartografía avanzada y la flexibilidad: relacional, objeto-relacional tipo de datos y XML
- Optimizado para un rendimiento altamente escalable y concurrencia con amplias opciones de ajuste del rendimiento
- Objeto de ayuda integral caché incluida la integración del clúster para algunos servidores de aplicaciones (por ejemplo, Oracle Fusion Middleware Server).
- Amplia capacidad de consulta que incluye: Java Persistence Query Language (JPQL), nativo de SQL, y el marco EclipseLink Expresiones.
- Objeto apoyo a nivel de transacción y la integración con servidores de aplicaciones y bases de datos populares.

Arquitectura

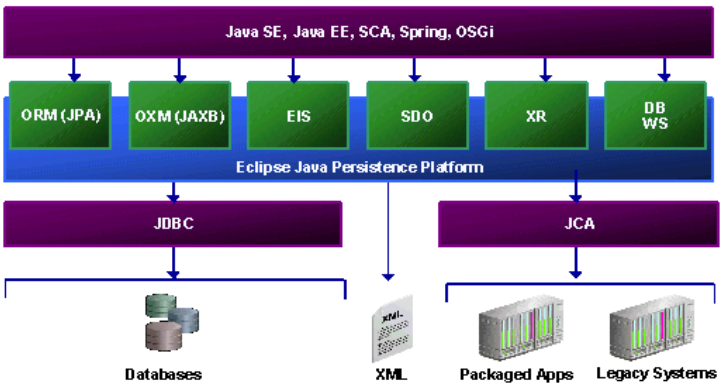


Figura IV.15. Arquitectura de Eclipselink

Fuente: http://www.jroller.com/afuentes/entry/proyecto_eclipselink

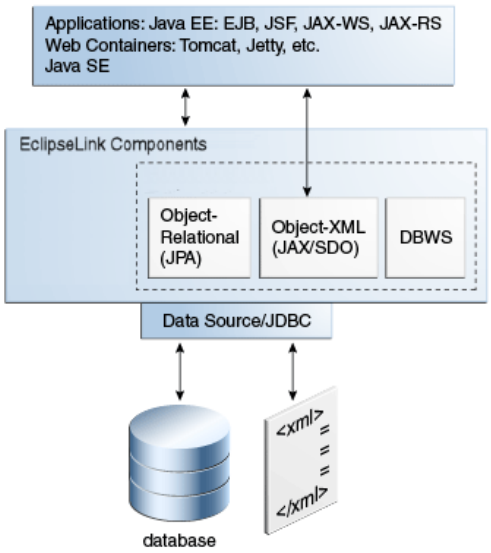


Figura IV.16. Componentes de Eclipselink

Fuente: http://www.jroller.com/afuentes/entry/proyecto_eclipselink

Componentes de EclipseLink

- **EclipseLink Core**

El núcleo EclipseLink proporciona el componente de tiempo de ejecución EclipseLink. El acceso al componente de tiempo de ejecución se puede obtener directamente a través de la API EclipseLink. El entorno de tiempo de ejecución no es un proceso independiente o externo que se inserta dentro de la aplicación. La aplicación llama a invocar EclipseLink para proporcionar un comportamiento de persistencia. Esta función permite el acceso transaccional y seguro para subprocesos de conexiones de bases de datos compartidas y objetos almacenados en caché.

- **EclipseLink / EclipseLink API**

El componente EclipseLink API de EclipseLink proporciona la implementación de referencia de JPA 2.0 (JSR-317). El org.eclipse.persistence.*classes encapsula la API EclipseLink y proporcionar extensiones más allá de la especificación. Estas extensiones incluyen propiedades específicas y anotaciones de Eclipselink.

- **Componente Objeto-Relacional (JPA 2.0)**

JPA proporciona un enfoque de mapeo objeto-relacional que permite definir de forma declarativa como mapear objetos Java a tablas de bases de datos relacionales de una manera estándar y portátil. APP funciona tanto en el interior de un servidor de aplicaciones Java EE y fuera de un contenedor EJB en una Java Standard Edition (Java SE) de la aplicación.

Las principales características incluidas en la actualización 2.0 APP son: Objeto Ampliado / funcionalidad de mapeo relacional

- Apoyo a las colecciones de objetos incrustados
- Múltiples niveles de objetos incrustados
- Las listas ordenadas
- Las combinaciones de tipos de acceso
- Una API de criterios de consulta
- Normalización de consulta "sugerencias"
- La normalización de metadatos adicionales para apoyar la generación de DDL
- Apoyo para la validación

- **Objetos XML (JAXB 2,2)**

Objeto-XML, también conocido como Moxy, es un componente EclipseLink que permite enlazar las clases Java a esquemas XML. Objetos XML implementa JAXB que le permite proporcionar información cartográfica a través de anotaciones. Soporte para almacenar las asignaciones en formato XML la proporciona Moxy. Las asignaciones muchas avanzadas que están disponibles le permiten manejar complejas estructuras XML sin tener que duplicar el esquema en el modelo de clases Java.

Los objetos producidos por el compilador JAXB EclipseLink son modelos Java POJO. Se generan con las anotaciones necesarias requeridas por la especificación JAXB.

Al utilizar objetos XML como proveedor JAXB, no hay metadatos se requiere para convertir su modelo de objetos existentes en XML. Puede proporcionar metadatos (con

anotaciones o XML) sólo cuando es necesario ajustar la representación XML del modelo.

Usando EclipseLink objetos XML, puede manipular XML de la siguiente manera:

- Generar un modelo de Java a partir de un esquema XML
- Especifique el tiempo de ejecución JAXB EclipseLink Moxy
- Usar JAXB para manipular XML
- Generar un esquema XML a partir de un modelo Java

- **EclipseLink SDO**

Los Service Data Objects (SDO) proporcionan la implementación de referencia de servicio de datos Objetos versión 2.1.1. La implementación de referencia se describe en la especificación JSR-235. La implementación EclipseLink SDO incorpora la implementación de referencia y proporciona características adicionales utilizadas principalmente para convertir objetos Java a XML, y para la construcción y el uso de modelos de objeto de datos que se pueden incorporar en las arquitecturas de servicios.

EclipseLink SDO le proporciona las siguientes capacidades:

- El uso de la API de SDO
- Conversión de un esquema XML
- Personalización de su XSD para el uso de SDO
- El uso de objetos de datos dinámicos para manipular XML
- El uso de objetos de datos estáticos
- Ejecute las SDO genera el compilador objetos de tipo de datos seguros
- Usar objetos de tipo de datos de seguridad para manipular XML

- **Web Services Database**

EclipseLink base de datos de Web Services (DBWS) permite un acceso sencillo y eficiente a los artefactos de bases de datos relacionales mediante el uso de un servicio

web. Proporciona Java EE que cumple como un cliente neutral acceso a la base de datos sin tener que escribir código Java.

EclipseLink DBWS amplía las capacidades básicas de EclipseLink durante el uso de ORM existentes y los componentes OXM.

EclipseLink DBWS tiene un componente de tiempo de ejecución de proveedor que tiene un descriptor de servicio (junto con los artefactos de implementación relacionadas) y se da cuenta como un servicio JAX-WS 2.0 Web. El proveedor de tiempo de ejecución utiliza EclipseLink de puente entre la base de datos y los mensajes SOAP XML utilizados por los clientes de servicios web.

Ventajas

- La información de asignación se puede almacenar externamente y no requiere ningún cambio en las clases de Java o de esquema XML. Esto significa que usted puede asignar los objetos de dominio de más de un esquema, o si los cambios de esquema, puede actualizar los metadatos de mapeo en lugar de modificar las clases de dominio. Esto también es útil cuando se asignan clases de terceros, porque no se puede tener acceso a la fuente para agregar anotaciones.
- Aborda ladisparidadentre los objetos Javay fuentes de datos.
- EclipseLinkes unFramework de persistenciaque le permite crearaplicaciones que combinanlos mejores aspectos dela tecnología de objetoscon una fuentede datos específica.
- La persistencia de objetosJava a prácticamentecualquier base de datosrelacional
- Realizaren memoriaconversionesentre los objetosJavay XML ydocumentos JSON

- Mapea cualquiermodelo de objetos paracualquieresquema relacionalono relacional.

Desventajas

- No proporciona sincronización de la caché entre la agrupación de las aplicaciones, la política de validaciones y la caché de consultas.
- La documentación referente a la definición de los elementos que contiene el Framework no es mucha.

4.4. DETERMINACIÓN DE LOS PARÁMETROS DE COMPARACIÓN.

El objetivo de este punto es determinar los parámetros que nos servirán para comparar los dos Frameworks de persistencia más importantes para el desarrollo de aplicaciones web Hibernate y Eclipselink, y establecer pautas que faciliten la elección a la hora de decidir cuál es el Framework más adecuado para la implementación del sistema.

4.4.1. PARÁMETRO 1: MAPEO OBJETO RELACIONAL

Este parámetro trata sobre las características principales que debe poseer un Framework de persistencia en cuanto al mapeo objeto relacional para convertir datos entre el sistema de tipos utilizando en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.

4.4.2. PARÁMETRO 2: PRODUCTO

En este parámetro se va a analizar las características más importantes propias de los dos Frameworks considerados y que son independientes del mecanismo, importante al momento de la elección de la herramienta para evitar posibles problemas en la implementación del sistema.

4.4.3. PARÁMETRO 3: MECANISMO

Con este parámetro se consideran aquellos aspectos que pueden aportar a la diferenciación entre los productos en este caso Hibernate y Eclipselink.

4.4.4. PARÁMETRO 4: COMPORTAMIENTO

El presente parámetro trata sobre las características de eficiencia de los Frameworks a analizar dentro de un aplicativo destacando su rendimiento, seguridad y otros aspectos fundamentales que ayudaran a establecer diferencias entre estos dos Frameworks.

4.5. DETERMINACIÓN DE LAS VARIABLES PARA LOS PARÁMETROS DE COMPARACIÓN

En la siguiente tabla tenemos las variables que se usaran en el análisis.

Tabla IV.VI. Tabla de Variables e indicadores a comparar

VARIABLE	INDICADORES
MAPEO OBJETO-RELACIONAL	Construcción manual de código para generar CRUD
	Creación Automática de Esquema
	DBMs Soportados
	Lenguaje de Consulta OO
	Apoyo de Cache
PRODUCTO	Documentación
	Última Versión
	Modo de Licenciamiento
	Integración con otros Frameworks

	Curva de Aprendizaje
RENDIMIENTO	Tiempo de Respuesta de consultas
	Uso de memoria
	Uso de CPU
COMPORTAMIENTO	Configuración del Framework
	Conversión de tipos
	Líneas de código generadas
	Internacionalización y localización

Fuente: Autor

4.6. ANÁLISIS COMPARATIVO

En esta sección se va a mostrar el estudio del FrameworkHibernate bajo la perspectiva de Framework de persistencia para el desarrollo de aplicaciones empresariales J2EE, la comprobación se hará por medio de cuadros comparativos en los que se va a determinar las variables de cada uno de los parámetros tomando en cuenta para el respectivo análisis, seguidos de una valoración, interpretación y calificación del criterio evaluado por parte del autor.

Para obtener resultados cuantitativos y cualitativos que permitan una selección adecuada de uno de los Frameworks analizados para desarrollar aplicaciones web empresariales, se realizara la calificación de cada uno de los parámetros de comparación que se basa en la siguiente escala:

Tabla IV.VII. Escala de calificación para parámetros de comparación

REGULAR	BUENO	MUY BUENO	EXCELENTE
< 70%	>=70% y <80%	>=80% y <90%	>=90%

Fuente: Autor

La siguiente tabla permitirá saber la valoración cuantitativa para las variables de los parámetros seleccionados:

Tabla IV.VIII. Escala de valoración Cuantitativa

1	2	3
No se cumple	Se cumple parcialmente	Se cumple
Malo	Bueno	Excelente
Deficiente	Poco eficiente	Muy eficiente
Nada	Poco	Mucho
No	Más o menos	Si
Empezando	Estable	Maduro

Fuente: Autor

Donde cada una de las variables van a ser evaluadas sobre el máximo que es **3** y cada uno de los ítems de la interpretación incluye la siguiente nomenclatura (x, y)/w en donde cada letra significa lo siguiente:

Tabla IV.IX. Significado de nomenclatura

x	**Representa el puntaje que obtiene el Framework Eclipselink**
y	Representa el puntaje que obtiene el FrameworkHibernate
w	Representa la base del puntaje sobre la cual se está calificando el parámetro

Fuente: Autor

La calificación definitiva de la herramienta en base a cada parámetro de comparación se obtiene sumando los puntajes obtenidos del análisis, utilizando las siguientes formulas:

$$P_{eclipselink} = \sum(x),\ P_{hibernate} = \sum(y),\ P_c = \sum(w)$$

Calificación de Eclipselink: $(C_c - eclipselin\,k) = (P_{eclipselink} / P_c) * 100\%$

Calificación de Hibernate: $(C_c - hibernate) = (P_{hibernate} / P_c) * 100\%$

En donde:

Tabla IV.X. Significado de variables

$P_{eclipselink}$	**Puntaje acumulado por Eclipselink en el parámetro analizado**
$P_{hibernate}$	Puntaje acumulado por Hibernate en el parámetro analizado
P_c	Puntaje sobre el que se califica el parámetro analizado
$C_c - eclipselin\,k$	Porcentaje de la calificación total que obtuvo Eclipselink en el parámetro
$C_c - hibernate$	Porcentaje de la calificación total que obtuvo Hibernate en el parámetro

Fuente: Autor

4.6.1. PARÁMETRO 1: MAPEO OBJETO-RELACIONAL

Las siguientes variables han sido escogidas en función de la capacidad de los 2 Frameworks evaluados, parámetros que deberán estar presentes tanto en Eclipselink y Hibernate.

4.6.1.1. Determinación de las variables

a. Construcción manual de código para generar CRUD
b. Creación Automática de Esquema
c. DBMs Soportados
d. Lenguaje de Consulta OO

e. Apoyo de Cache

- **Valorizaciones**

A continuación se describe las consideraciones a evaluar en cada uno de las variables ya determinadas en el parámetro producto.

a) Variable Construcción manual de código para generar CRUD

Se evaluará si con alguno de los dos Frameworks el desarrollador puede crear sus propios métodos ya sea con sentencias SQL o con su propio lenguaje de consulta para acceder y manipular los datos.

b) Variable Creación Automática de Esquema

Se refiere a si el producto provee la funcionalidad de generar de forma automática el esquema de base de datos al que se mapean los objetos persistentes.

c) Variable DBMs Soportados

Este parámetro refiere al conjunto de las base de datos que son soportadas por el mapeador en cuestión.

d) Variable Lenguaje de Consulta orientado a objetos

Indica si el mapeador en cuestión provee un lenguaje deconsulta orientado a objetos.

e) Variable Apoyo de cache

Refiere a si el mapeador permite especificar la propagación de operaciones de persistencia a lo largo de sus asociaciones.

Tabla IV.XI. Resumen variables parámetro Mapeo objeto-relacional

VARIABLE	Ecipselink	Hibernate
Construcción manual de código para generar CRUD	Poco eficiente	Muy Eficiente
CreaciónAutomática de Esquema	Excelente	Excelente
DBMs soportados	Se cumple	Se cumple

	Parcialmente	
Lenguaje de Consulta orientado a objetos	Poco eficiente	Muy Eficiente
Apoyo de Cache	Bueno	Excelente

Fuente: Autor

4.6.1.2. Interpretación de la valorización

Las escalas definidas en las tablas IV.VIII y IV.IX, se han aplicado a las variables valorizadas como se indican en la tabla IV.XI, interpretándolas de la siguiente manera.

- La construcciónmanual de código para generar CRUD, es permitida en el caso de Hibernate siendo este no muy invasivo caso contrario Eclipselink que crea tanto los controladores como métodos de cada clase son creados automáticamente lo que genera código que no es tan mantenible. **(2,3)/3**

- Tanto Hibernate como Ecipselink proporcionan la funcionalidad de generación de esquema, en el caso de Hibernate es posible activar la generación de esquemas utilizando la configuración de las propiedades Hibernate.hb2dl.auto, que se puede ajustar a uno de los valores siguientes: validar, actualizar, crear y crear abandono, entonces antes de que se cree EntityManagerFactory Hibernate genera sentencias DDL y las ejecuta en la base de datos.

  Al igual que Eclipselink donde la generación de esquemas se puede activar mediante persistence.xml aquí también existen opciones de crear instrucciones DDL para que se ejecuten en la base de datos. **(3,3)/3**

- **DBMs Soportados.**

Tabla IV.XII. DBMs Soportados por Hibernate

N°	RDBMS	Dialecto
1	DB2	org.Hibernate.dialect.DB2Dialect
2	DB2 AS/400	org.Hibernate.dialect.DB2400Dialect
3	DB2 OS390	org.Hibernate.dialect.DB2390Dialect
4	PostgreSQL	org.Hibernate.dialect.PostgreSQLDialect
5	MySQL5	org.Hibernate.dialect.MySQL5Dialect
6	MySQL5 with InnoDB	org.Hibernate.dialect.MySQL5InnoDBDialect
7	MySQL con MyISAM	org.Hibernate.dialect.MySQLMyISAMDialect
8	Oracle (cualquier versión)	org.Hibernate.dialect.OracleDialect
9	Oracle 9i	org.Hibernate.dialect.Oracle9iDialect
10	Oracle 10g	org.Hibernate.dialect.Oracle10gDialect
11	Oracle 11g	org.Hibernate.dialect.Oracle10gDialect
12	Sybase	org.Hibernate.dialect.SybaseASE15Dialect
13	Sybase Anywhere	org.Hibernate.dialect.SybaseAnywhereDialect
14	Microsoft SQL Server 2000	org.Hibernate.dialect.SQLServerDialect
15	Microsoft SQL Server 2005	org.Hibernate.dialect.SQLServer2005Dialect
16	Microsoft SQL Server 2008	org.Hibernate.dialect.SQLServer2008Dialect
17	SAP DB	org.Hibernate.dialect.SAPDBDialect
18	Informix	org.Hibernate.dialect.InformixDialect
19	HypersonicSQL	org.Hibernate.dialect.HSQLDialect
20	H2 Database	org.Hibernate.dialect.H2Dialect

N°	RDBMS	Dialecto
21	Ingres	org.Hibernate.dialect.IngresDialect
22	Progress	org.Hibernate.dialect.ProgressDialect
23	Mckoi SQL	org.Hibernate.dialect.MckoiDialect
24	Interbase	org.Hibernate.dialect.InterbaseDialect
25	Pointbase	org.Hibernate.dialect.PointbaseDialect
26	FrontBase	org.Hibernate.dialect.FrontbaseDialect
27	Firebird	org.Hibernate.dialect.FirebirdDialect

Fuente:http://www.jpab.org/Hibernate/MySQL/server/EclipseLink/MySQL/server.html

Tabla IV.XIII. DBMs Soportados por Eclipselink

N°	DATABASE	JAVA CLASS
1	Apache Derby	Org.eclipse.persistence.platform.database.DerbyPlatform
2	Attunity	Org.eclipse.persistence.platform.database.AttunituPlatform
3	DBase	Org.eclipse.persistence.platform.database.DBasePlatform
4	Firebird	Org.eclipse.persistence.platform.database.FirebirdPlatform
5	H2	Org.eclipse.persistence.platform.database.H2Platform
6	HyperSQL	Org.eclipse.persistence.platform.database.HSQLPlatform
7	IBM DB2	Org.eclipse.persistence.platform.database.DB2Platform
8	Informix	Org.eclipse.persistence.platform.database.InformixPlatform
9	Microsoft Access	Org.eclipse.persistence.platform.database.AccessPlatform
10	SQL Server	Org.eclipse.persistence.platform.database.SQLServerPlatform
11	MySQL	Org.eclipse.persistence.platform.database.MySQLPlatform
12	Oracle	Org.eclipse.persistence.platform.database.OraclePlatform

13	PostgreSQL	Org.eclipse.persistence.platform.database.PostgreSQLPlatform
14	PointBase	Org.eclipse.persistence.platform.database.PoinBasePlatform
15	SAP DB	Org.eclipse.persistence.platform.database.MaxDBPlatform
16	Sybase	Org.eclipse.persistence.platform.database.SybasePlatform

Fuente: http://www.jpab.org/Hibernate/MySQL/server/EclipseLink/MySQL/server.html

- Como se puede observar en las tablas anteriores Hibernate soporta mayor cantidad de DBMS en este caso 27 caso contrario pasa con Eclipselink que soporta solamente 16 bases de datos lo que es un punto favor a Hibernate que podría ser utilizado con la gran mayoría de plataformas DBMS. **(2,3)/3**

- Hibernate posee HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación en el caso de Eclipselink posee JPQL (Java Persistence Query Language) un lenguaje propio de creación de consultas muy similar a SQL. Se crean las consultas preparadas (similar a las Prepared Statement) asociadas a cada entidad mediante anotaciones del tipo, pero adicional a esto Hibernate soporta sentencias SQL y pose QBE y QBC. **(2,3)/3**

- En resumen las soluciones de almacenamiento en caché en implementaciones, Hibernate intenta delegar el almacenamiento en caché a herramientas especializadas, mientras que EclipseLink integra el almacenamiento en caché de segundo nivel en el núcleo del marco de trabajo. La ventaja innegable del enfoque de Hibernate es que no trata de recuperar toda la colección es por ello que es un Frameworks de almacenamiento en caché ya maduro y sofisticado.

  (2,3)/3

➢ **Calificación**

$$P_c = \sum(w) = 3+3+3+3+3 = 15$$

$$P_{eclipselink} = \sum(x) = 2+3+2+2+2 = 11$$

$$C_c - eclipselin\,k : (P_{eclipselink} / P_c) * 100\% = (11/15) * 100\% = 73{,}33\%$$

$$P_{hibernate} = \sum(y) = 3+3+3+3+3 = 15$$

$$C_c - hibernate : (P_{hibernate} / P_c) * 100\% = (15/15) * 100\% = 100\%$$

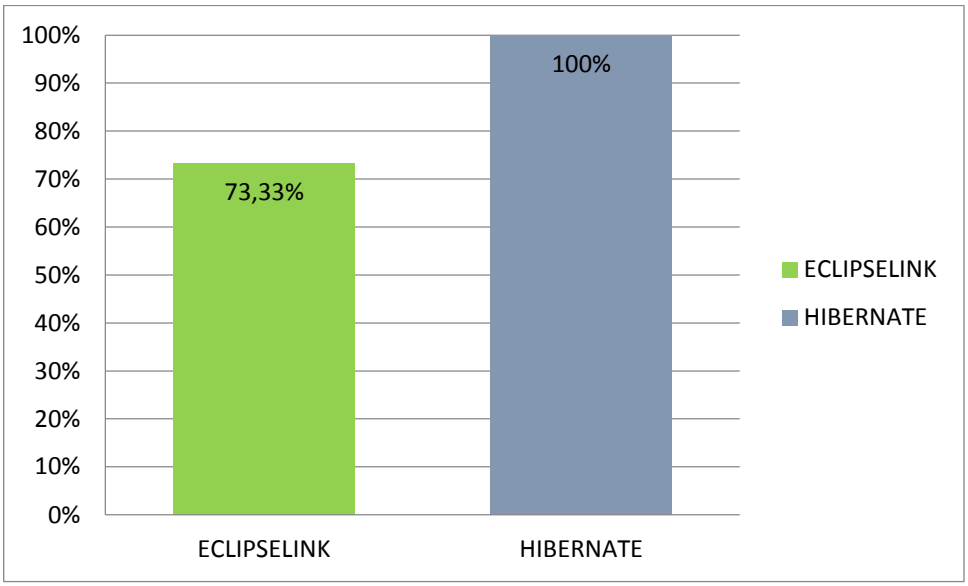


Figura IV.17.Comparación estadística del parámetro Mapeo Objeto Relacional

Fuente: Autor

4.6.2. PARÁMETRO 2: PRODUCTO

4.6.2.1.Determinación de las variables

a. Documentación

b. Última Versión

c. Modo de Licenciamiento

d. Integración con otros Frameworks

e. Curva de Aprendizaje

- **Valorizaciones**

A continuación se describe las consideraciones a evaluar en cada uno de las variables ya determinadas en el parámetro producto.

a) Variable Documentación

Esta variable permitir conocer si existe la información suficiente de los Frameworks, tanto de Hibernate como de Eclipselink, es claro que la disponibilidad de documentación es un factor muy importante para la comprensión y aprendizaje.

b) Variable Última Versión

Se refiere a la última versión estable del producto. Este criterio es de interés debido a que la versión es un indicador de la madurez y estabilidad del mismo.

c) Variable Modo de Licenciamiento

Esta variable permite conocer la licencia bajo la cual se distribuyen los Frameworks analizados.

d) Integración con otros Frameworks

Esta variable permitirá definir si los Frameworks analizados tienen compatibilidad con otrosFrameworksque ampliaríansus capacidades aún más.

e) Variable Curva deAprendizaje

Indica el esfuerzo requerido para la comprensión y aprendizaje dela forma de uso del producto. Esta medida es importante en lo que refiere a la capacitación de los desarrolladores en el caso de no tener experiencia con el producto.

Tabla IV.XIV. Resumen variables parámetro Producto

VARIABLE	ECLIPSELINK	HIBERNATE
Documentación	Buena	Excelente
Última versión	Estable	Maduro
Modo de licenciamiento	Bueno	Excelente
Integración con otros Frameworks	Bueno	Excelente
Curva de Aprendizaje	Mediana	Corta

Fuente: Autor

4.6.2.2. Interpretación de la valorización

Las escalas definidas en las tablas IV.VIII y IV.IX, se han aplicado a las variables valorizadas como se indican en la tabla IV.XIV, interpretándolas de la siguiente manera.

- Documentación, durante mi trabajo enesta tesisfueampliamentela lectura de la documentación sobrecadatema en particulardiscutidoaquí y, por lo tanto, soyelegible para proporcionaruna comparacióndelacalidad de la documentacióndelos Frameworks.

  Hibernatetiene una documentación ampliade alta calidad de cada nueva versión, hay documentosque describen suscaracterísticas, es posibleencontrar documentacióndelas versiones anterioresy todo se explica claramenteconejemplos. Puesto que Hibernate es el más utilizado en aplicaciones existe una gran cantidad deejemplostodos a través de Internety los usuarios puedenhacer preguntas endiscusiónforos.

  Al contrario de EclipseLink, donde hay una gran cantidad de documentación, pero su problemaes la fragmentación. De hecho, esposible encontrarrespuestas ala mayoría de laspreguntas, pero toma bastante tiempoen comparacióndelas Hibernate.

Además, dado quela documentación no es por versiones, a veces es bastante difícil distinguir qué funcionalidad es nueva. **(2,3)/3**

- En cuanto a la última versión de Hibernate se encuentra en la 4.1.3 lo que muestra un grado alto de madurez es una herramienta ORM completa que ha conseguido en un tiempo record una excelente reputación en la comunidad de desarrollo, al contrario de Eclipselink que se deriva de Oracle Eclipselink es un producto nuevo en el ambiente de persistencia se encuentra en la versión en prueba 2.4. **(2,3)/3**
- Ambos Frameworks afortunadamente son libres bajo licencias permisivas y de negocios. Hibernate utiliza LGPL y Eclipselink es doble licenciado bajo la licencia publica de eclipse. **(3,3)/3**
- Integración con otros Frameworks, Hibernate es conocido por su excelente integración con diversos Frameworks para el desarrollo de aplicaciones web ya sean Frameworks de almacenamiento de cache como Ehcahe o Hazelcast o también con Spring, Struts, en cuanto a Eclipselink también permite la integración con otros Frameworks JEE. **(3,3)/3**
- Curva de aprendizaje en el caso de Hibernate es muy corta, fácil de aprender y sencillo de utilizar, lo opuesto a Eclipse link por el hecho de falta de documentación por ser una tecnología nueva. **(2,3)/3**

- **Calificación**

$$P_c = \sum(w) = 3+3+3+3+3 = 15$$

$$P_{eclipselink} = \sum(x) = 2+2+3+3+2 = 12$$

$$C_c - eclipselin\,k : (P_{eclipselink} / P_c) * 100\% = (12/15) * 100\% = 80\%$$

$$P_{hibernate} = \sum(y) = 3+3+3+3+3 = 15$$

$$C_c - hibernate : (P_{hibernate} / P_c) * 100\% = (15/15) * 100\% = 100\%$$

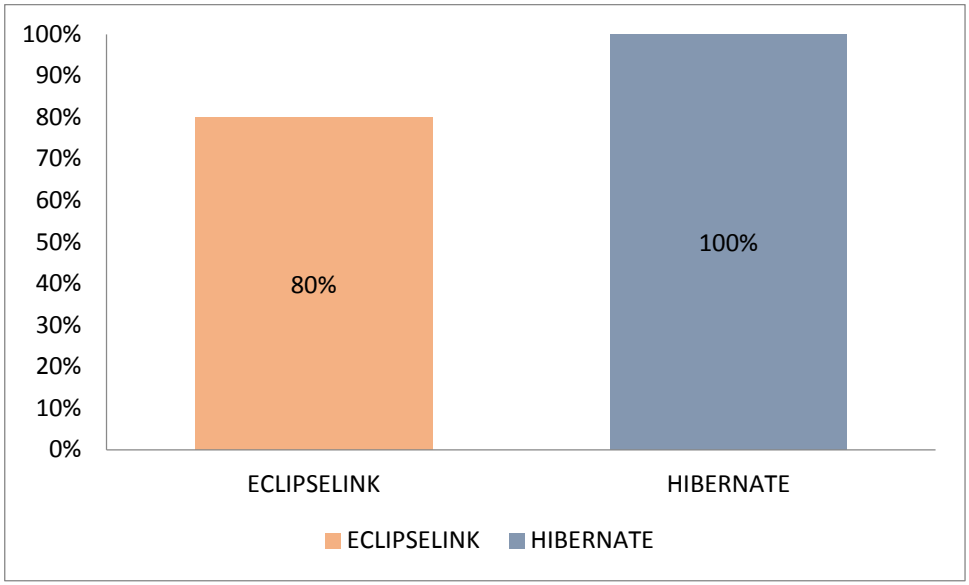


Figura IV.18.Comparación estadística del parámetro Producto

Fuente: Autor

4.6.3. PARÁMETRO 3: RENDIMIENTO

En este grupo no se incluyen todos los criterios considerados a nivel de mecanismos, Sino que solo se consideran aquellos que pueden aportar a la diferenciación de los Frameworks analizados.

4.6.3.1 Determinación de las variables

a. Tiempo de respuesta de consulta
b. Uso de memoria
c. Uso de CPU

- **Valorizaciones**

A continuación se describe las consideraciones a evaluar en cada uno de las variables ya determinadas en el parámetro producto.

a) Tiempo de Respuesta

Una característica muy importante dentro del rendimiento de cada Framework, describe el tiempo en que se demora en realizar una consulta hasta generar su respuesta.

b) Uso de Memoria

Se refiere a la cantidad de memoria ocupada por el Framework al momento de realizar una consulta.

c) Uso de CPU

Se refiere a la cantidad de CPU ocupada por el Framework al momento de realizar una consulta.

Para poder valorizar e interpretar cada variable este parámetro se realizó una prueba mediante el desarrollo de dos aplicaciones una con Hibernate y la otra con Eclipselink, con el fin de comparar las características de rendimiento, en estas pruebas se van a determinar los tiempos de respuesta, registros afectados, uso de memoria y uso de procesador al momento de ejecutar sentencias CRUD. Cabe indicar que se realizaron consultas SQLque implican uniones de tablas con el fin de descubrirla eficacia con las dostecnologías en comparación Con este propósito se crearon las aplicaciones en los siguientes escenarios.

Tabla IV.XV. Escenario 1. Hibernate

Ítem	**Descripción**
HARDWARE	
Procesador	Intel Core i5 2.5 Ghz
Memoria Ram	4 GB
Disco Duro	600 GB
Cache	2.5 g Ghz
SOFTWARE	

Sistema Operativo	Windows 7 Home Premium
IDE	Netbeans
Base de Datos	Postgre SQL
Framework de persistencia	Hibernate

Fuente: Autor

Tabla IV.XVI. Escenario 2. Eclipselink

Ítem	**Descripción**
HARDWARE	
Procesador	Intel Core i5 2.5 Ghz
Memoria Ram	4 GB
Disco Duro	600 GB
Cache	2.5 g Ghz
SOFTWARE	
Sistema Operativo	Windows 7 Home Premium
IDE	Netbeans
Base de Datos	Postgre SQL
Framework de persistencia	Eclipselink

Fuente: Autor

Posterior al desarrollo de los aplicativos en las siguientes figuras se presentan los resultados obtenidos de las pruebas realizadas.

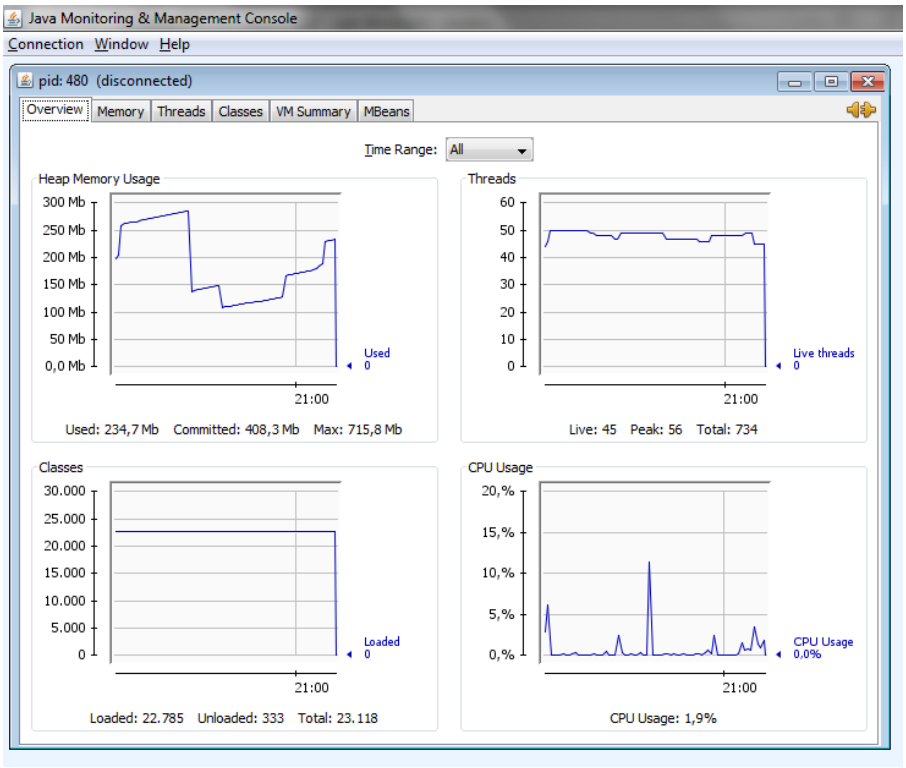


Figura IV.19.Test de Rendimiento de Eclipselink

Fuente: Autor

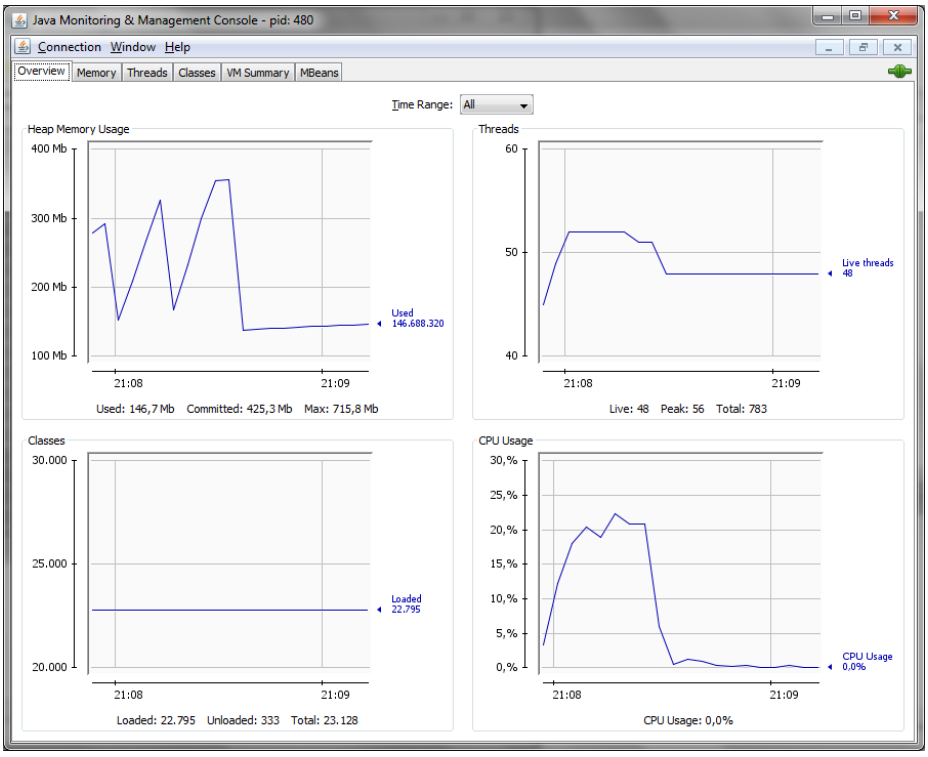


Figura IV.20. Test de Rendimiento de Hibernate

Fuente: Autor

Tabla IV.XVII. Resumen variables parámetro Rendimiento

VARIABLE	ECLIPSELINK	HIBERNATE
Tiempo de respuesta de consulta	Poco eficiente	Muy eficiente
Uso de Memoria	Poco eficiente	Muy Eficiente
Uso de CPU	Poco eficiente	Muy Eficiente

Fuente: Autor

4.6.3.2. Interpretación de la valorización

Las escalas definidas en las tablas IV.VIII y IV.IX, se han aplicado a las variables valorizadas como se indican en la tabla IV.XVII, interpretándolas de la siguiente manera.

- En lo relacionado al tiempo de respuesta, encontramos que Hibernate supero a Eclipselink en la mayoría de pruebas realizadas, pese a que Ecipselink tuvo menor tiempo en la primera vez que se ejecutaron las aplicaciones en las posteriores no fue así esto se dio porque Hibernate administra de mejor manera la memoria cache de segundo nivel contrario a Eclipselink, también se puedo observar que Hibernate ejecuta mejor las consultas en donde existen tablas relacionadas en las pruebas se pudo observar que en la inserción de 10003 tres registros se demoró 4 min con 30s caso contrario a Hibernate que se demoró 28s. **(2,3)/3**
- Al analizar en las gráficas de rendimiento de los dos Frameworks se observa que la cantidad de memoria requerida por Hibernate es menor a la ocupada por Eclipselink. **(2,3)/3**
- En cuanto al uso de la CPU se comprobó que Hibernate utiliza mayor porcentaje pero esto se ve recompensado a que termina pronto las operaciones CRUD y procede a liberar este recurso caso contrario con Eclipselink que utiliza menor cantidad de este pero por mayor tiempo. **(2,3)/3**

- **Calificación**

$$P_c = \sum(w) = 3+3+3 = 9$$

$$P_{eclipselink} = \sum(x) = 2+2+2 = 6$$

$$C_c - eclipselin\,k : (P_{eclipselink} / P_c) * 100\% = (6/9) * 100\% = 66{,}7\%$$

$$P_{hibernate} = \sum(y) = 3+3+3 = 9$$

$$C_c - hibernate : (P_{hibernate} / P_c) * 100\% = (9/9) * 100\% = 100\%$$

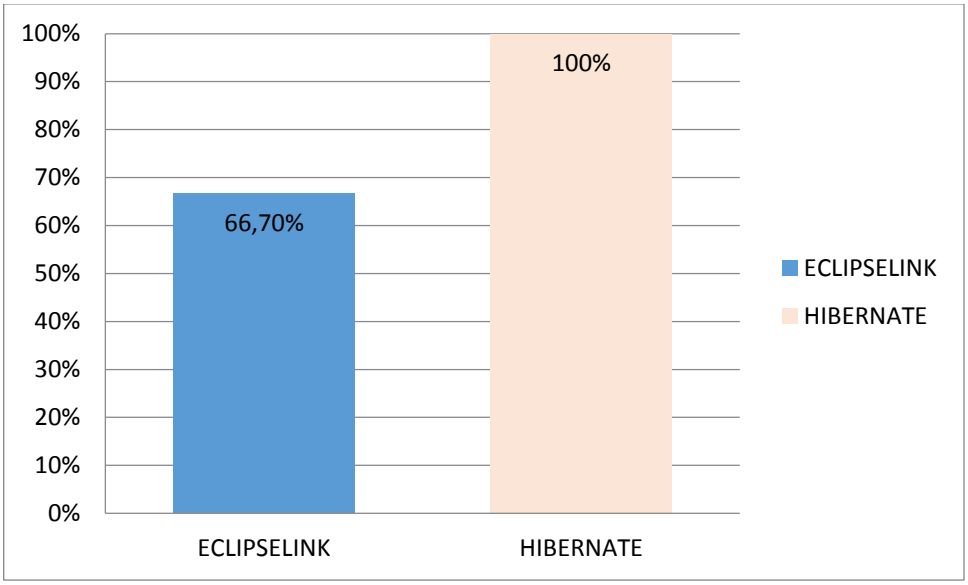


Figura IV.21.Comparación estadística del parámetro Rendimiento

Fuente: Autor

4.6.4. PARÁMETRO 4: COMPORTAMIENTO

Las siguientes variables han sido escogidas en función de la conducta de los 2 Frameworks evaluados, parámetros que deberán estar presentes tanto en Eclipselink y Hibernate.

4.6.4.1. Determinación de las variables

a. Configuración del Framework
b. Conversión de tipos
c. Líneas de código generadas
d. Internacionalización y localización

- **Valorizaciones**

A continuación se describe las consideraciones a evaluar en cada uno de las variables ya determinadas en el parámetro comportamiento.

a) Variable Configuración del Framework

La configuración previa de un Framework es el paso inicial para su funcionamiento, se evaluara si los Frameworks ofrecen una configuración transparente y el grado de dificultad que presenta.

b) Conversión de tipos

Se relaciona a la manera de como los Frameworks analizados convierten los tipos de datos desde las tablas creadas en la base de datos a las clases java generadas.

c) Líneas de código generadas

Su significado varía de un lenguaje de programación a otro, pero también dentro de un mismo lenguaje de programación. Este indicador permitir a evaluar la cantidad de líneas de código que se llevan para realizar un determinado proceso con Hibernate y Eclipselink, con la finalidad de establecer el Framework que meno líneas de código genera.

d) Internacionalización y localización

Esta variable permitirá conocer el soporte para internacionalización tanto de Eclipselink como de Hibernate.

Tabla IV.XVIII. Resumen variables parámetro Comportamiento

VARIABLE	ECLIPSELINK	HIBERNATE
Configuración del Framework	Excelente	Excelente
Conversión de tipos	Muy eficiente	Muy eficiente
Líneas de código generadas	Poco eficiente	Poco eficiente
Internacionalización y localización	Buena	Muy Buena

Fuente: Autor

4.6.4.2. Interpretación de la valorización

Las escalas definidas en las tablas IV.VIIIy IV.IX, se han aplicado a las variables valorizadas como se indican en la tabla IV.XVIII, interpretándolas de la siguiente manera.

- La configuración previa para usar el Framework es de extrema importancia, en este caso ambos Frameworks poseen archivos de configuración XML, en donde se configura la conexión a la base de datos y generan el modelo de dominio.

  (3,3)/3

- En lo referente a la conversión de tipos Hibernate existe la interfaz type donde un objeto Type Hibernate hace corresponder un tipo Java con un tipo de una columna de la base de datos. Todas las propiedades persistentes de las clases persistentes, incluyendo las asociaciones, tienen un tipo Hibernate correspondiente. Este diseño hace que Hibernate sea altamente flexible y extendible. Incluso se permiten tipos definidos por el usuario (interfaz UserType y CompositeUserType), por otro lado en EclipseLink, la interfaz primaria para la definición de convertidores personalizadoses Converter. Las aplicaciones pueden crear implementaciones personalizadasde esta interfaz o utilizar algunos de los convertidores predefinidos desdela distribución EclipseLink.

  (3,3)/3

- La cantidad de líneas de código generadas por Hibernate es un poco mayor a la de las líneas generadas por Eclipselink. **(3,2)/3**
- En el caso de Hibernate la internacionalización llega a estar cubierto por el JBoss Portal en su contendedor de portales a través del uso de recursos en los porlets en el caso de Eclipselink en la actualidad es buena aunque no sea muy conocido y no sea tan reconocido como Hibernate. **(2,3)/3**

- **Calificación**

$$P_c = \sum(w) = 3+3+3+3 = 12$$

$$P_{eclipselink} = \sum(x) = 3+3+3+2 = 11$$

$$C_c - eclipselin\,k : (P_{eclipselink} / P_c) * 100\% = (11/12) * 100\% = 91{,}7\%$$

$$P_{hibernate} = \sum(y) = 3+3+2+3 = 11$$

$$C_c - hibernate : (P_{hibernate} / P_c) * 100\% = (11/12) * 100\% = 91{,}7\%$$

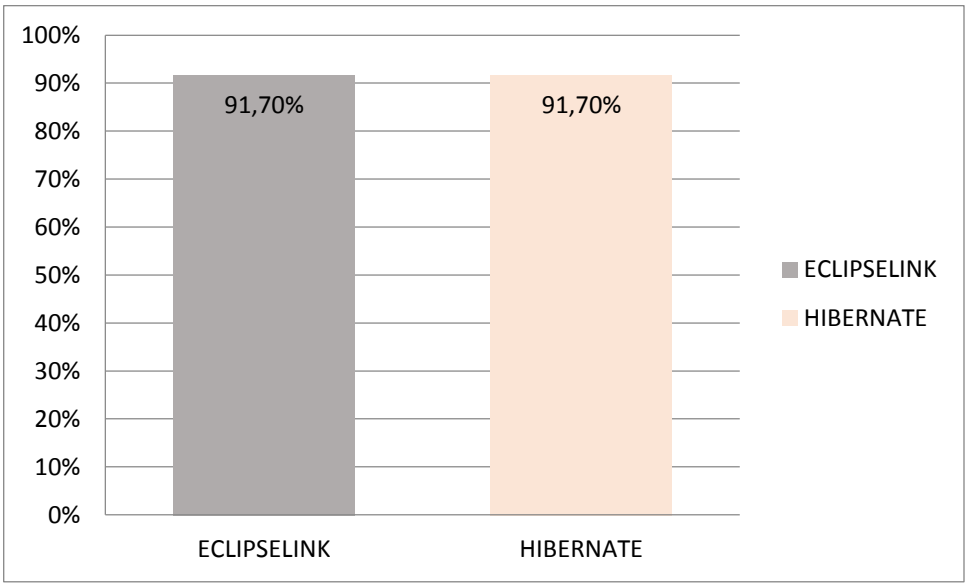


Figura IV.22.Comparación estadística del parámetro Comportamiento

Fuente: Autor

4.7 ANÁLISIS DE LOS RESULTADOS

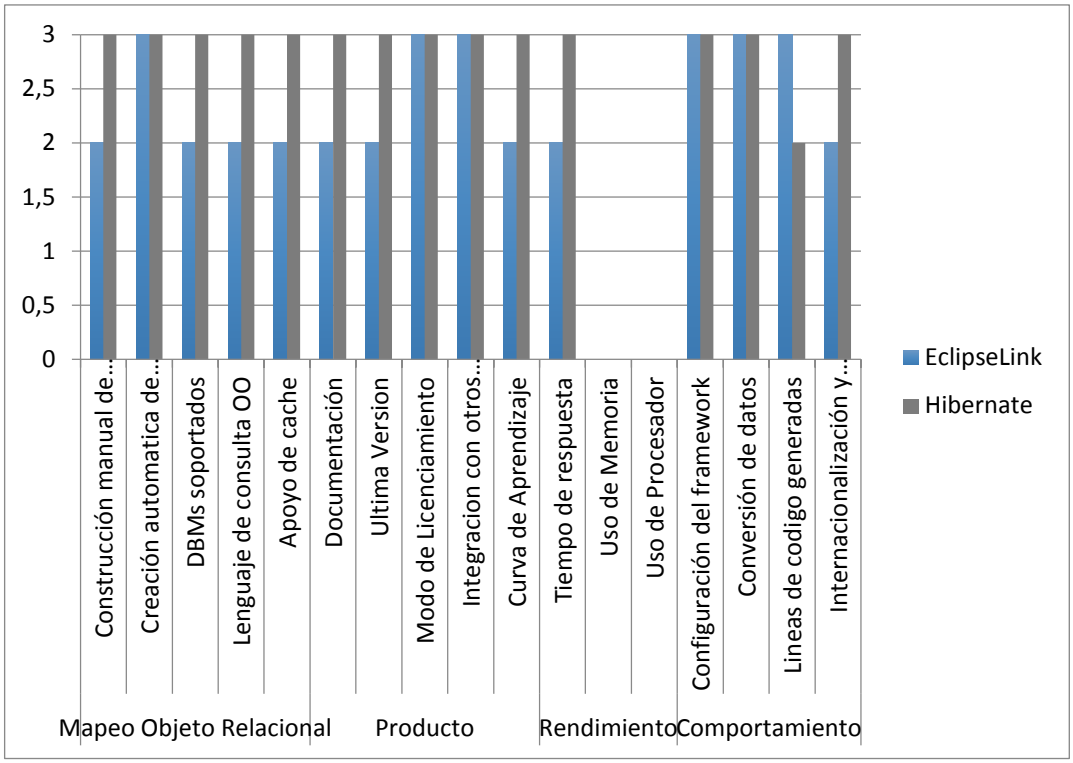


Figura IV.23. Resultado Final por parámetro

Fuente: Autor

$P_T = 15+15+9+12 = 51$

$PT_{eclipselink} = 11+12+6+11 = 40$

$PT_{hibernate} = 15+15+9+11 = 50$

$(\%Eclipselink) = (40/51)*100 = 78{,}43\%$ **Equivalente a Bueno**

$(\%Hibernate) = (50/51)*100 = 98\%$ **Equivalente a Excelente**

Resultados que podemos representar de la siguiente manera:

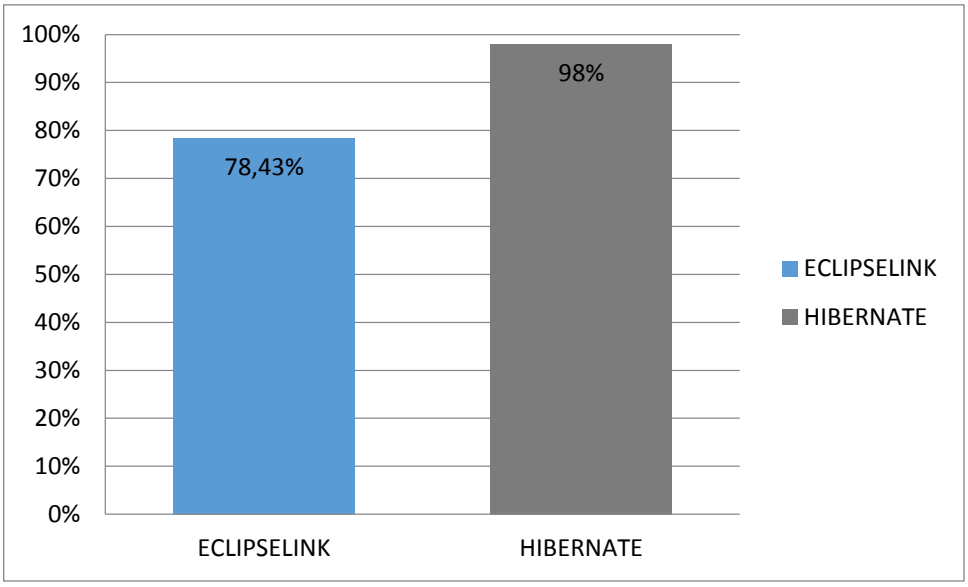


Figura IV.24.Diagrama general de resultados

Fuente: Autor

Después de haber realizado el análisis comparativo correspondiente de los respectivos parámetros con sus variables, como se puede observar en la Figura IV.11 se ha determinado que para desarrollar una aplicación web empresarial sobre la arquitectura J2EE, el FrameworkHibernate es el que tiene una máxima calificación de 98% correspondiente a excelente por lo que se determina que es el framework más adecuado para desarrollar la aplicación web empresarial en ambiente J2EE; por otro lado Eclipselink obtuvo una calificación de 78,43% que equivale a bueno.

4.8 DEMOSTRACIÓN DE LA HIPÓTESIS

Para poder realizar la demostración de la hipótesis de este trabajo investigativo se va a probar que con el uso del FrameworkHibernate mejorará el control y uso de los bienes internos y externos del Gobierno Municipal de Carlos Julio Arosemena Tola.

Se tomara en consideración dos escenarios el primero es como se realizaba el control y uso de los bienes anteriormente sin la existencia de algún sistema y el segundo será con la implementación del sistema de control de bienes.

4.8.1. MODELO PARA LA COMPROBACIÓN DE LA HIPÓTESIS

Tabla IV.XIX. Definición de Variables

Variable	Tipo Variable	Concepto
Desarrollo de una aplicación web basada en el FrameworkHibernate	Independiente Compleja	Aquellas aplicaciones que los usuarios pueden utilizar accediendo a un Servidor web a través de Internet o de una intranet mediante un navegador.
Mejorar el control y uso de bienes	Dependiente Compleja	Procesos adecuados que permitan orientar los recursos a los lugares correctos en el menor tiempo posible.

Fuente: Autor

Tabla IV.XX. Definición de Indicadores

Variable	Categoría	Indicadores	Técnica	Fuentes de Investigación
Mejorar el control y uso de bienes	Calidad de servicio	Grado de satisfacción de los usuarios Disponibilidad de información	Observación Encuesta	GAD MUNICIPAL DE AROSEMENA TOLA
	Rendimiento	Tiempo de respuesta a	Observación Encuesta	GAD MUNICIPAL

		requerimientos Costo		DE AROSEMENA TOLA

Fuente: Autor

4.8.2. PLANTEAMIENTO DE LA HIPÓTESIS

“El desarrollo de una aplicación web, basada en el Framework Hibernate como apoyo al nivel Administrativo mejorará el control y uso de los bienes internos y externos del Gobierno Municipal”

Hipótesis Nula (Ho)= La implementación de una aplicación web basada en Hibernate nomejorará el control y uso de bienes internos y externos del Gobierno Municipal. A>B

Hipótesis Alternativa (Hi)=La implementación de una aplicación web basada en Hibernate mejorará el control y uso de bienes internos y externos del Gobierno Municipal. A<=B

4.8.3. SELECCIÓN DE NIVEL DE SIGNIFICANCIA.

El nivel de significancia es de 0,05 %, que es aplicado a proyectos de investigación.

4.8.4. DESCRIPCION DE LA MUESTRA.

La muestra para nuestra investigación es finita, aplicada a los administradores de los departamentos del GAD Arosemena Tola.

4.8.5. ESPECIFICACIÓN DEL ESTADÍSTICO

El tipo de distribución sedeterminó por el tamaño de la muestra, que en mi trabajo de investigación es de 12 personas, por lo que se debe aplicar la Distribución T-Student.

$$t = \frac{x - \mu}{\sigma / \sqrt{n-1}} \text{ cuando } n < 30$$

4.8.6. ESPECIFICACION DE LAS REGIONES DE RECHAZO Y ACEPTACIÓN.

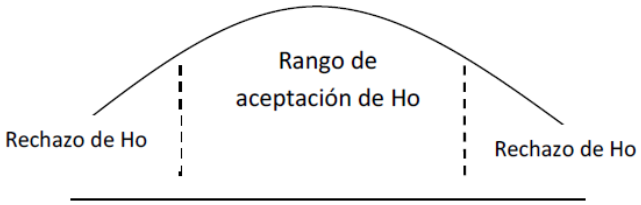


Figura IV.25.Regiones de rechazo y aceptación

Fuente: Autor

4.8.7. RECOLECCION DE DATOS Y CÁLCULOS DE LOS ESTADÍSTICOS

Tabla IV.XXI. Valoración

	DESCRIPCION		VALORIZACION
No	Pésimo	Nunca	0
	Malo	Casi nunca	1
	Bueno	A menudo	2
	Satisfactorio	Muy a menudo	3
Si	Muy Satisfactorio	Siempre	4

Fuente: Autor

4.8.8. TABULACION DE LA INFORMACIÓN

De las encuestas realizadas a 12 personas que son los encargados de realizar los procesos de gestión de bienes se obtuvo los siguientes resultados.

SIN EL SISTEMA

Pregunta 1: ¿La institución cuenta con un Sistema de Solicitud y Control de Bienes que interactúe con todos los departamentos?

Tabla IV.XXII. Resultados Pregunta 1 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Si	0	0
No	12	0
	Total	0

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 2: ¿Cuál es el grado de satisfacción con la forma manual con la que se viene llevando el control y uso de bienes?

Tabla IV.XXIII. Resultados Pregunta 2 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Muy Satisfactorio	0	0
Satisfactorio	0	0
Bueno	0	0
Malo	10	10
Pésimo	2	0
	Total	10

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 3: ¿Cómo calificaría el proceso de solicitud de materiales a bodega en la actualidad?

Tabla IV.XXIV. Resultados Pregunte 3 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Muy Satisfactorio	0	0
Satisfactorio	0	0
Bueno	4	8
Malo	6	6
Pésimo	2	0
	Total	14

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 4: ¿Cómo calificaría usted la calidad de información sobre la disponibilidad de materiales dada por los responsables de bodega?

Tabla IV.XXV. Resultados Pregunta 4 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Muy Satisfactoria	0	0
Satisfactoria	0	0
Buena	4	8
Mala	5	5
Pésima	3	0
	Total	13

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 5: ¿Usted conoce la disponibilidad de los materiales de una obra antes de generar una orden de entrega?

Tabla IV.XXVI. Resultados Pregunta 5 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Siempre	0	0
Muy a menudo	0	0
A menudo	2	4
Casi nunca	8	1
Nunca	2	0
	Total	5

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 6: ¿El Departamento tiene conocimiento de las obras que se desarrollan en el cantón?

Tabla IV.XXVII. Resultados Pregunta 6 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Si	10	40
No	2	0
	Total	40

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 7: ¿Cómo califica el proceso de entrega y control de combustibles en la institución?

TABLA IV.XXVIII. RESULTADOS PREGUNTA 7 SIN NINGÚN SISTEMA

RESPUESTAS	CANTIDAD	VALORACION
Muy satisfactorio	0	0
Satisfactorio	0	0
Bueno	1	2

Malo	8	8
Pésimo	3	0
	Total	10

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 8: ¿Existe información disponible sobre los equipos de cómputo para fines de administración y toma de decisiones?

Tabla IV.XXIX. Resultados Pregunta 8 sin ningún sistema

RESPUESTAS	CANTIDAD	VALORACION
Si	2	8
No	10	0
	Total	8

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

La valoración total para la forma de manejo actual en control y uso de bienes es de 100 sobre 384 puntos que es el puntaje máximo que podría obtenerse en la encuesta realizada.

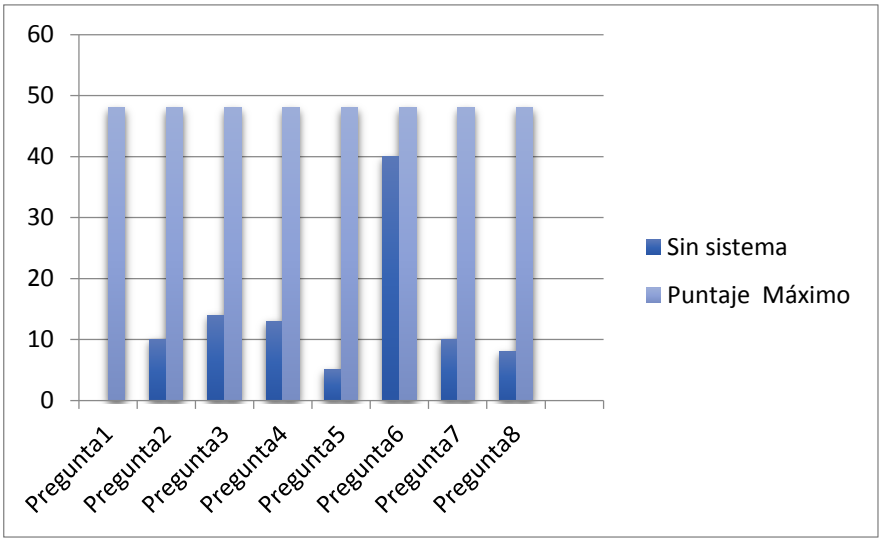


Figura IV.26.Resultado encuesta situación actual.

Fuente: Autor

$$Porcentaje = \frac{100*100}{384} = 26.04\%$$

Como resultado se obtiene que el manejo mediante la forma manual alcanza unporcentaje de 26.04% de control, uso de bienes y representa el grado de satisfacción del usuario.

SISTEMA SGB

Pregunta 1: ¿La institución cuenta con un Sistema de Solicitud y Control de Bienes que interactúe con todos los departamentos?

Tabla IV.XXX. Resultados pregunta 1 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Si	12	48
No	0	0
	Total	48

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 2: ¿Cuál es el grado de satisfacción con el funcionamiento del sistema SGB desarrollado?

Tabla IV.XXXI. Resultados Pregunta 2 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Muy Satisfactorio	10	40
Satisfactorio	2	6
Bueno	0	0
Malo	0	0
Pésimo	0	0
	Total	46

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 3: ¿Cómo calificaría el proceso de solicitud de materiales a bodega en la actualidad?

Tabla IV.XXXII. Resultados Pregunta 3 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Muy Satisfactorio	10	40
Satisfactorio	2	6
Bueno	0	0
Malo	0	0
Pésimo	0	0
	Total	46

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 4: ¿Cómo calificaría usted la calidad de información sobre la disponibilidad de materiales dada por los responsables de bodega?

Tabla IV.XXXIII. Resultados Pregunta 4 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Muy Satisfactoria	11	44
Satisfactoria	1	3
Buena	0	0
Mala	0	0
Pésima	0	0
	Total	47

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 5: ¿Usted conoce la disponibilidad de los materiales de una obra antes de generar una orden de entrega?

Tabla IV.XXXIV. Resultados Pregunta 5 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Siempre	12	48
Muy a menudo	0	0
A menudo	0	0
Casi nunca	0	0
Nunca	0	0
	Total	48

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 6: ¿El Departamento tiene conocimiento de las obras que se desarrollan en el cantón?

Tabla IV.XXXV. Resultados Pregunta 6 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Si	12	48
No	0	0
	Total	48

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 7: ¿Cómo califica el proceso de entrega y control de combustibles en la institución?

Tabla IV.XXXVI. Resultados Pregunta 7 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Muy satisfactorio	10	40
Satisfactorio	2	6
Bueno	0	0

Malo	0	0
Pésimo	0	0
	Total	46

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

Pregunta 8: ¿Existe información disponible sobre los equipos de cómputo para fines de administración y toma de decisiones?

Tabla IV.XXXVII. Resultados Pregunta 8 para el sistema SGB

RESPUESTAS	CANTIDAD	VALORACION
Si	12	48
No	0	0
	Total	48

Fuente: Autor

Valorada sobre 48 puntos que es el máximo puntaje posible para esta pregunta.

La valoración total para la forma de manejo actual en control y uso de bienes es de 369 sobre 384 puntos que es el puntaje máximo que podría obtenerse en la encuesta realizada.

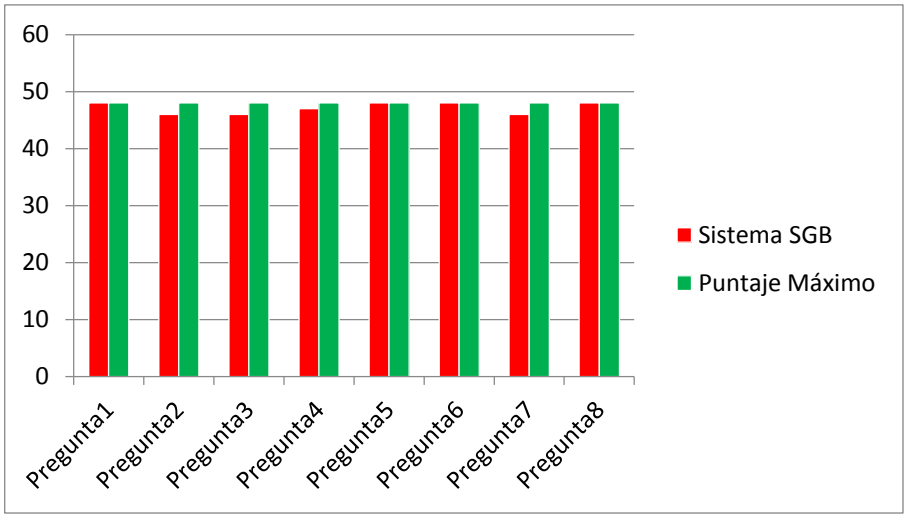


Figura IV.27.Resultado encuesta Sistema SGB

Fuente: Autor

$$Porcentaje = \frac{369*100}{384} = 96{,}09\%$$

Como resultado se obtiene que el manejo mediante la forma manual alcanza un porcentaje de 26.38% de control, uso de bienes y representa el grado de satisfacción del usuario.

La siguiente tabla muestra los resultados de la valoración total para cada pregunta en los sistemas analizados.

Tabla IV.XXXVIII. Resultado final

Encuestas	Sin sistema	Sistema SGB
1	10	31
2	14	31
3	13	32
4	9	31
5	8	32
6	8	32
7	9	31
8	9	31
9	7	32
10	5	32
11	8	31
12	7	31

Fuente: Autor

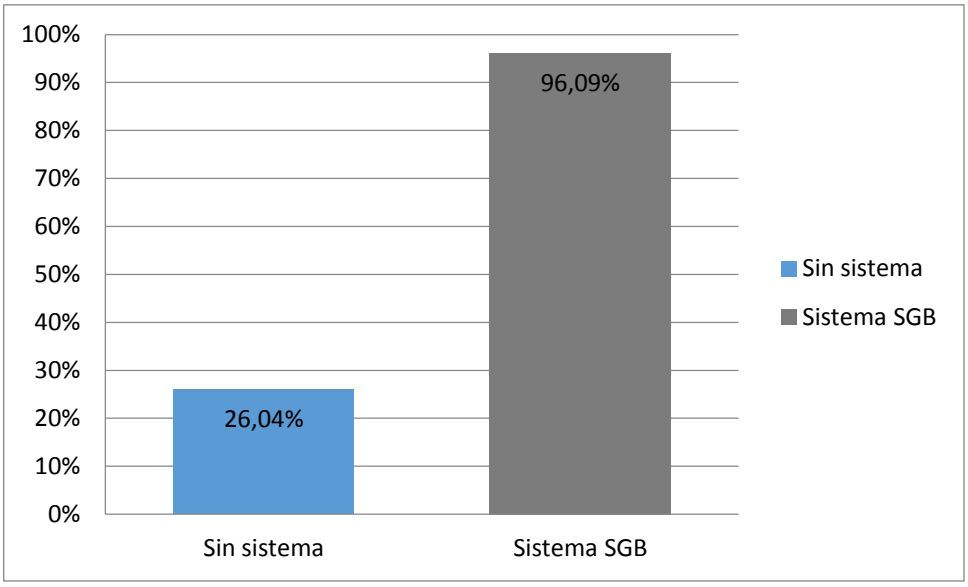


Figura IV.28. Resultado final

Fuente: Autor

Tenemos que obtener las medias, desviación estándar y la varianza para realizar los posteriores cálculos.

Medias

$$\bar{x}_1 = \frac{10+14+13+9+8+8+9+9+7+5+8+7}{12} = 8{,}92$$ **Sin Sistema**

$$\bar{x}_2 = \frac{31+31+32+31+32+31+31+31+32+32+31++31}{12} = 31{,}42$$ **SGB**

Varianza

Sin sistema

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

$$s^2 = \frac{\sum_{i=1}^{12}(x_i - \bar{x})^2}{12-1}$$

$$s^2 = \frac{(10-8.92)^2 + (14-8.92)^2 + (13-8.92)^2 + (9-8.92)^2 + (8-8.92)^2 + (8-8.92)^2 + (9-8.92)^2 + (9-8.92)^2 + (7-8.92)^2 + (5-8.92)^2 + (8-8.92)^2 + (7-8.92)^2}{12-1}$$

$$s^2 = \frac{1.17 + 25.81 + 16.65 + 0.0064 + 0.85 + 0.85 + 0.0064 + 0.0064 + 3.69 + 15.37 + 0.85 + 3.69}{11}$$

$$s^2 = \frac{68.95}{11}$$

$$s^2 = 6.27$$

Sistema SGB

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

$$s^2 = \frac{\sum_{i=1}^{12}(x_i - \bar{x})^2}{12-1}$$

$$s^2 = \frac{(31-31.42)^2 + (31-31.42)^2 + (32-31.42)^2 + (31-31.42)^2 + (32-31.42)^2 + (32-31.42)^2 + (31-31.42)^2 + (31-31.42)^2 + (32-31.42)^2 + (32-31.42)^2 + (31-31.42)^2 + (31-31.42)^2}{12-1}$$

$$s^2 = \frac{0.18 + 0.18 + 0.34 + 0.18 + 0.34 + 0.34 + 0.18 + 0.18 + 0.34 + 0.34 + 0.18 + 0.18}{11}$$

$$s^2 = \frac{2.96}{11}$$

$$s^2 = 0.27$$

Desviación estándar

$$s = \sqrt{6.27} = 2.5$$

$$s = \sqrt{0.27} = 0.52$$

$$s^2{}_p = \frac{(n_1-1)s^2{}_1 + (n_2-1)s^2{}_2}{n_1 + n_2 - 2} = \frac{11*6.27 + 11*0.27}{12+12+2} = 3.27$$

$$t = \frac{\bar{x}_1 - \bar{x}_2}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} = \frac{8.92 - 31.42}{1.81\sqrt{\frac{1}{11} + \frac{1}{11}}} = -29.22$$

Tabla IV.XXXIX. CálculosEstadísticos

Encuestas	Sin sistema	Sistema SGB	d	$d-D$	$(d-D)^2$
1	10	31	-21	1.5	2.25
2	14	31	-17	5.5	30.25
3	13	32	-19	3.5	12.25
4	9	31	-22	0.5	0.25
5	8	32	-24	-1.5	2.25
6	8	32	-24	-1.5	2.25
7	9	31	-22	0.5	0.25
8	9	31	-22	0.5	0.25
9	7	32	-25	-2.5	6.25
10	5	32	-27	-4.5	20.25
11	8	31	-23	-0.5	0.25
12	7	31	-24	-1.5	2.25
			$\sum d = 270$		$\sum (d-D)^2 = 79$

Fuente: Autor

Calculo de la prueba estadística

$$\sigma d = \frac{\sqrt{\sum (d-D)^2}}{n-1} = \frac{\sqrt{79}}{12-1} = 0.81$$

$$t = \frac{D}{\frac{\sigma d}{\sqrt{n}}} = \frac{-22.5}{0.23} = -97.83$$

Por otro lado, $t_{\frac{\alpha}{2},n1+n2-2} = t_{0,0025;22} = 2,0739$

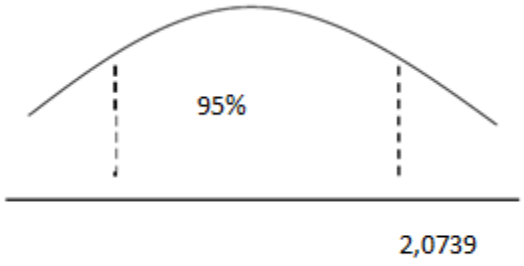


Figura IV.29. Zona de la Prueba estadística

Fuente: Autor

4.8.9 DECISION ESTADÍSTICA

Con un nivel de significancia del 0,05% y un análisis a dos colas, se observa que el valor mediante la prueba t cae en el rango de rechazo de la hipótesis nula Ho por lo que aceptamos la hipótesis alternativa Hi que dice “La implementación de una aplicación web basada en Hibernate mejorará el control y uso de bienes internos y externos del Gobierno Municipal”.

Interpretación: Una aplicación web basada en Hibernate en el GAD Municipal Arosemena Tola mejora el control y uso de bienes, existiendo diferencias significativas entre el antes (sin sistema) y el después (con el sistema SGB).

CAPÍTULO V

5. DISEÑO Y DESARROLLO DE LA APLICACIÓN

INTRODUCCIÓN

En el presente capítulo se pone en ejecución la parte aplicativa del proyecto, los módulos de la solución web para los procesos de Gestión de Bienes del Gobierno Autónomo Descentralizado de Carlos Julio Arosemena Tola, usando Hibernate como Framework de persistencia de Datos en la tecnología J2EE de Java.

5.1 MICROSOFT SOLUTION FRAMEWORK

5.1.1 DEFINICIÓN

El MSF es un modelo diseñado específicamente para crear productos de muy buena calidad, donde para cumplir este objetivo prima la comunicación tanto entre el equipo de desarrollo como entre ellos y los clientes

5.1.2 FASES

El MSF está compuesto por las siguientes fases:

1. Visión
2. Planeación
3. Desarrollo
4. Estabilización
5. Instalación
6. Soporte

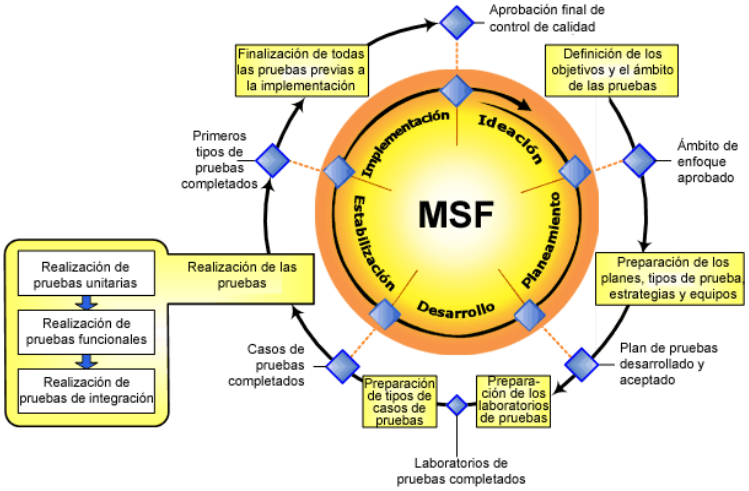


Figura V.30. Fases de MSF

Fuente: http://technet.microsoft.com/es-es/library/bb490314.aspx

5.1.3 VENTAJAS

La ventaja principal es que al ser un modelo desarrollado por Microsoft se puede tener mayor soporte y mantenimiento, además la mayoría de los usuarios finales están más acostumbrados con este producto. Además sirve para grandes y pequeños proyectos. Cabe recalcar que MSF no se parece al RUP en algunas definiciones (principalmente en la cuestión de los cambios).

- Incentiva al trabajo en equipo y la colaboración
- Es útil para proyectos de pequeña y gran escala.

- Crea una disciplina de análisis de riesgos que ayudan a evolucionar con el proyecto.
- Cuenta con plantillas que nos ayuda para el proceso de documentos.

5.1.4 DESVENTAJAS

La principal desventaja es que se torna un trabajo bastante largo, ya que para cada fase se debe documentar profundamente todo lo que se haga, pero no deja de ser un modelo que tiene buenos resultados

5.1.5 FASE DE VISIÓN

Para el desarrollo eficiente del proyecto es importante obtener una visión del proyecto compartida, comunicada, extendida y alineada con los objetivos del negocio. Además identificar los beneficios, requerimientos funcionales, sus alcances y restricciones; y los riesgos inherentes al proceso

5.1.6 DEFINICIÓN DEL PROBLEMA

La falta de automatización de ciertos procesos en la gestión de bienes identificados por el nivel estratégico de la institución, han ocasionado la falta de seguimiento y control de los bienes.

- La falta de información de los recursos disponibles de una obra dificulta el trabajo de asignación de materiales, genera desvíos de los mismos y conflictos al contabilizar
- La falta de control sobre el uso de combustible genera gastos excesivos a la institución

- La contraloría solicito a la institución implementar un mecanismo para tener el historial de cada equipo de cómputo de la Institución por lo que se implementó ese modulo en el sistema

5.1.7 VISIÓN DEL PROYECTO

Desarrollar un sistema que permita a los departamentos responsables de las obras contar con la disponibilidad de los materiales adquiridos para una determinada obra, antes de generar una orden de entrega.

Proveer información detallada sobre los materiales adquiridos para una obra durante y después de finalizar las obras.

Informar al nivel administrativo responsable sobre los kilómetros recorridos y rendimiento de los vehículos con la cantidad de combustible entregado, de tal manera que sea una herramienta para tomar decisiones.

Registrar el historial de una máquina de tal manera que pueda hacer un seguimiento sobre los equipos de cómputo de la institución.

- **Módulo de Administración y Sistemas**

Este módulo consta de varios módulos y permiten crear procesos para que pueda funcionar y trabajar los usuarios de los departamentos, Bodega, Combustible.

  - Módulo de Administración de Usuarios
  - Módulo de Departamentos
  - Módulo de Representantes
  - Módulo de Asignación de recursos
  - Módulo de Administrar Computadoras
  - Módulo de Reportes

- **Módulo de Departamento**
  - Módulo de Solicitudes de Compras

- Módulo de Solicitudes de Entrega
- Módulo de Representantes
- Módulo de Lugares
- Módulo de Obras
- Módulo de reportes

- **Módulo de Bodega**
  - Módulo de Solicitud de Compras
  - Módulo de Solicitud de Entrega
  - Módulo de Proveedores
  - Módulo de Categorías
  - Módulo de Unidades de medida
  - Módulo de Productos
  - Módulo de Reportes
- **Módulo de Combustibles**
  - Módulo de Compras
  - Módulo de Entrega
  - Módulo de Productos
  - Módulo de Vehículos
  - Módulo de Proveedores
  - Módulo de Reportes
- **Módulo de Seguridad**

El modulo permitirá realizar la autenticación de nuestros usuarios a través de un usuario y contraseña, además de darle un tipo que le es asignado por el administrador.

- Módulo de Usuarios

5.2 PERFILES DE USUARIO

Al realizar el análisis de los usuarios para el sistema se ha determinado los siguientes: Usuario Súper Administrador (crea el ambiente para el funcionamiento del sistema, jefe del área de sistemas), Usuario Administrador (responsable de los departamentos) Usuario invitado (Alcalde)

Tabla V.XL. Perfiles de Usuario

No.	Nombre	Tipo	Tipo de Acceso	Descripción
1	Súper Administrador	Usuario Administrador	Web	Es el encargado de crear a los usuarios, el ambiente de trabajo para los departamentos
2	Administrador	Usuario Administrador	Web	Es el encardo de solicitar Compras, solicitar entrega de materiales, crear obras, Beneficiarios, lugares, tienen control total sobre el área de trabajo por lo que se ha limitado al acceso a recursos.
3	Invitado	Usuario Invitado	Web	Acceso a reportes

Fuente: Autor

5.3 ÁMBITO DEL PROYECTO

Desarrollo de una aplicación web para la Gestión y Control de bienes del GAD Municipal de Arosemena Tola.

Este sistema permitirá la Gestión y Administración de los módulos incorporados que permiten realizar la gestión y control de bienes.

5.3.1 REQUERIMIENTOS FUNCIONALES

Módulo de Administración de Usuarios

Req1. El sistema deberá permitir gestionar datos de usuarios

Módulo de Departamentos

Req2. El sistema deberá permitir gestionar datos de los departamentos

Módulo de Representantes

Req3. El sistema deberá permitir gestionar datos de los Representantes

Módulo de Asignación de Recursos

Re4. El sistema deberá permitir asignar datos de los recursos

Módulo de Administrar Computadoras

Req5. El sistema deberá permitir gestionar datos de las partes

Req6. El sistema deberá permitir gestionar datos de los mantenimientos

Módulo de Compras (bodega)

Req7. El sistema deberá permitir gestionar los datos de las órdenes de compras

Req8. El sistema deberá permitir gestionar los ítems de las órdenes de compra.

Módulo de Solicitud de Entrega (bodega)

Req9. El sistema deberá permitir gestionar los datos de las solicitudes de entrega realizadas por los departamentos

Req10. El sistema deberá permitir gestionar los ítems de las solicitudes de entrega.

Módulo de Proveedores

Req11. El sistema deberá permitir gestionar los datos de los proveedores

Módulo de Categorías

Req12. El sistema deberá permitir gestionar los datos de las categorías

Módulo de Unidades de medida

Req13. El sistema deberá permitir gestionar los datos de las unidades de medida

Módulo de Productos

Req14. El sistema deberá permitir gestionar los datos de los productos

Módulo de Solicitudes de Compras (Departamentos)

Req15. El sistema deberá permitir gestionar los datos para solicitudes de compra a Bodega

Módulo de Solicitudes de Entrega (Departamentos)

Req16. El sistema deberá permitir gestionar datos para solicitudes de entrega a Bodega

Módulo de Lugares

Req17. El sistema deberá permitir gestionar datos de los lugares

Módulo de Obras

Req18. El sistema deberá permitir gestionar datos de las obras

Módulo de Vehículos

Req19. El sistema deberá permitir gestionar datos de los vehículos

Módulo de entrega Combustible

Req20. El sistema deberá permitir registrar datos de las órdenes de Entrega de combustible por parte de sección combustible.

Módulo de solicitud de entrega (Departamento)

Req21. El sistema deberá permitir registrar datos de las ordenes de generada por el departamento.

Módulo de Reportes

Req22. EL sistema debe generar reportes

- Lista de Departamentos
- Lista de Responsables
- Lista de Computadoras
- Lista de Lugares
- Lista de Obras

- Lista de Órdenes de Compra
- Lista de Ordenes de Entrega
- Lista de Materiales Disponibles de Una obra
- Lista de Materiales Entregados en una Obra
- Lista de Vehículos
- Lista de Unidades de Medida
- Lista de Categorías
- Lista de Rendimientos
- Lista de Mantenimientos
- Lista de Custodios

5.3.2 REQUERIMIENTOS NO FUNCIONALES

- Rendimiento
- Disponibilidad
- Seguridad
- Portabilidad
- Mantenibilidad
- Escalabilidad
- Reusabilidad
- Interfaces
- Usabilidad

5.4 OBJETIVO DEL PROYECTO

5.4.1 OBJETIVOS DEL NEGOCIO

- Registrar a los diferentes Usuarios, registrando su cuenta.
- Llevar un control transparente de los procesos de gestión y control de bienes
- Limitar el acceso autorizado solo a las personas autorizadas.
- Generación de reportes

5.4.2 OBJETIVOS DE DISEÑO

- Garantizar un acceso rápido a la aplicación.
- Proteger contra el acceso de intrusos mediante la validación de cuentas.
- Proporcionar una Interfaz web amigable y de fácil manejo

5.5 RIESGOS

El riesgo implica cambios que pueden darse por cambios de opinión, de acciones, de lugares, es inevitable e implica incertidumbre y pérdida cuando el riesgo se ha convertido en problema.

Debemos analizar qué riesgos podrían hacer que nuestro proyecto fracasara, y se debe escoger adecuadamente que acciones pueden convertirse en riesgos para de esta manera lograr superarlos.

El propósito esencial de este proceso de análisis de riesgos va enfocado a prevenir muchos acontecimientos que pueden dificultar el desarrollo del “Sistema de Gestión de Bienes”.

Por lo cual debemos darle una gran importancia a este análisis tomando en cuenta todos los tipos de riesgos que se puede encontrar en el desarrollo del sistema al mismo tiempo gestionar aquellos riesgos con gran probabilidad de impacto.

5.6 IDENTIFICACIÓN DEL RIESGO

La identificación del riesgo es un intento sistemático para especificar las amenazas al plan de proyecto (estimaciones, planificación temporal, carga de recursos, etc.). Identificando los riesgos conocidos y predecibles, el gestor del proyecto da un paso adelante para evitarlos cuando sea posible y controlarlos cuando sea necesario.

Existen tres tipos de riesgo:

- Riesgo del proyecto.
- Riesgo técnico.
- Riesgo del negocio

Tabla V.XLI. Identificación de Riesgos

ID	DESCRIPCION DEL RIESGO	CATEGORIA	CONSECUENCIA
R1	Los usuarios no definieron correctamente los requerimientos.	Proyecto	Retraso del proyecto, costo del proyecto.
R2	Cambios continuos de los requerimientos por parte del usuario	Negocio	Pérdida de recursos económicos.
R3	Falta de conocimiento por parte de los programadores del lenguaje de programación y desarrollo de software	Técnico	Amenazan la calidad Capacitación a los programadores Retraso del proyecto
R4	Costo final del proyecto sea exagerado	Proyecto	Retraso del proyecto, Costo del proyecto.
R5	Interfaces inadecuadas para validar las operaciones en el	Técnico	Amenazan la calidad del software y la implementación

	sistema.		puede llegar a ser difícil.
R6	Falta de comunicación con los miembros del municipio.	Proyecto	Retraso en el proyecto
R7	La información que se modifique no se actualice en la base de datos, del sistema	Negocio	Provocan inconsistencia en los datos

5.6.1 ANÁLISIS DE RIESGOS

Tabla V.XLII. Valoración de riesgos

Rango de Probabilidad	Descripción	Valor
1% - 33%	Baja	1
34% - 67%	Media	2
68% - 99%	Alta	3

Fuente: Autor

Tabla V.XLIII. Probabilidad

Identificación	Probabilidad		
	%	Valor	Probabilidad
R1	20	1	BAJA
R2	20	1	BAJA
R3	30	1	BAJA
R4	40	2	MEDIA
R5	20	1	BAJA
R6	50	1	MEDIA
R7	60	2	MEDIA

Fuente: Autor

Determinación del Impacto

Tabla V.XLIV. Impacto del Riesgo

Impacto	Retraso	Impacto técnico	Costo	valor
Bajo	1 semana	Ligero efecto en el desarrollo del proyecto	< 1%	1
Moderado	2 semanas	Moderado efecto en el desarrollo del proyecto	< 5%	2
Alto	1 mes	Severo efecto en el desarrollo del proyecto	< 10%	3
Crítico	> 1 meses	Proyecto no puede ser culminado	> 20%	4

Fuente: Autor

Tabla V.XLV. Riesgo-Impacto

Identificación	Impacto	
	Valor	**Impacto**
R1	3	ALTO
R2	3	ALTO
R3	1	BAJO
R4	3	ALTO
R5	1	BAJO
R6	2	MEDIA
R7	3	MEDIA

Fuente: Autor

Determinación de la exposición al riesgo

Tabla V.XLVI. Impacto-Probabilidad

Exposición al riesgo	Valor	color
Baja	1 o 2	
Media	3 o 4	
Alta	>6	

Fuente: Autor

Impacto / Probabilidad	Bajo =1	Moderado =2	Alto =3	Crítico =4
Alta = 3	3	6	9	12
Media = 2	2	4	6	8
Baja = 2	2	4	6	8

Fuente: Autor

Tabla V.XLVII. Tabla tolla Riesgo

Identificación	**Probabilidad**			**Impacto**		**Exposición al riesgo**	
	%	Valor	Probabilidad	Valor	Impacto		
R1	20	1	BAJA	3	ALTO	3	MEDIA
R2	20	1	BAJA	3	ALTO	3	MEDIA
R3	30	1	BAJA	1	BAJO	1	BAJA
R4	40	2	MEDIA	3	ALTO	6	ALTA
R5	20	1	BAJA	1	BAJO	1	BAJA
R6	50	1	MEDIA	2	MODERADO	2	BAJA
R7	60	2	MEDIA	3	ALTO	6	MEDIA

Fuente: Autor

Tabla V.XLVIII. Prioridades del riesgo

Identificación	**Prioridad**	**Exposición**
R4	1	6
R1	2	3
R2	2	3
R7	2	3
R6	3	2
R3	4	1

R5	4	1

Fuente: Autor

5.6.2 PLANEACIÓN Y PROGRAMACIÓN DEL RIESGO

Para mitigar el riesgo se utiliza la Hoja de Gestión de Riesgo

Tabla V.XLIX. Gestión del Riesgo 1

HOJA DE GESTIÓN DEL RIESGO			
ID DEL RIESGO: R1		**FECHA:**	
Probabilidad: Baja **Valor: 1**	**Impacto: Alto** **Valor: 3**	**Exposición: Alta** **Valor: 3**	**Prioridad: 1**
DESCRIPCIÓN: Los usuarios no definieron correctamente los requerimientos.			
REFINAMIENTO: **Causas:** No exista una comunicación adecuada entre el usuario y responsable del proyecto El Usuario no explico correctamente las necesidades que posee. **Consecuencias:** Retraso del proyecto Incremento en el costo del proyecto			
REDUCCIÓN: Analizar las necesidades del usuario para establecer correctamente los requerimientos. Que exista una adecuada comunicación entre el cliente y el programador			
SUPERVISIÓN:			

Ponerse de acuerdo al inicio: el cliente y el responsable acerca de sus necesidades. Que el ambiente de comunicación sea el más propicio entre el cliente y el responsable del proyecto.
GESTIÓN: Que una vez conocidas las necesidades del cliente se deberán poner de acuerdo el responsable del proyecto y cliente para establecer los requerimientos.
ESTADO ACTUAL: Fase de reducción iniciada: [x] Fase de supervisión iniciada: [ ] Gestionando el riesgo: [ ]
RESPONSABLE: Oljer Cando

Fuente: Autor

Tabla V.L. Gestión del Riesgo 2

HOJA DE GESTIÓN DEL RIESGO			
ID DEL RIESGO: R2		**FECHA:**	
Probabilidad: Baja **Valor: 1**	**Impacto: Alto** **Valor: 3**	**Exposición:** **Alta** **Valor: 3**	**Prioridad:** **1**
DESCRIPCIÓN: Cambios continuos de los requerimientos por parte del usuario			
REFINAMIENTO:			

Causas: Falta de análisis de requerimientos por parte del usuario y desarrollador **Consecuencias:** No cumplir con las expectativas del sistema Pérdida de recursos
REDUCCIÓN: Asignación de recursos para una correcta captación de los requerimientos
SUPERVISIÓN: Seguimiento de las funcionalidades de los requisitos planteados
GESTIÓN: Desplegar los beneficios de un requerimiento gestionado
ESTADO ACTUAL: Fase de reducción iniciada: [x] Fase de supervisión iniciada: [ ] Gestionando el riesgo: [ ]
RESPONSABLE: Oljer Cando

Fuente: Autor

Tabla V.LI. Gestión del Riesgo 3

<table>
<tr><th colspan="4">HOJA DE GESTIÓN DEL RIESGO</th></tr>
<tr><td colspan="2">ID DEL RIESGO: R3</td><td colspan="2">FECHA:</td></tr>
<tr><td>Probabilidad: Baja
Valor: 1</td><td>Impacto:
Baja
Valor: 1</td><td>Exposición:
Baja
Valor: 1</td><td>Prioridad: 4</td></tr>
<tr><td colspan="4">DESCRIPCIÓN: Falta de conocimiento por parte de los programadores del lenguaje de programación y desarrollo de software</td></tr>
<tr><td colspan="4">REFINAMIENTO:

Causas:
Especialización en diferentes lenguajes de programación y desarrollo del software al que se utilizará para la elaboración del nuevo sistema
Falta de investigación acerca del lenguaje de programación que se usará.
Consecuencias:
Amenaza de calidad
Capacitación a los programadores
Retraso del proyecto</td></tr>
<tr><td colspan="4">REDUCCIÓN:
Capacitar al programador en el lenguaje de programación y desarrollo de software que se usara
Promover la investigación de nuevos lenguajes de programación</td></tr>
<tr><td colspan="4">SUPERVISIÓN:</td></tr>
</table>

Controlar que la capacitación a los programadores sea la adecuada.
GESTIÓN: Fomentar la especialización en el lenguaje de programación
ESTADO ACTUAL: Fase de reducción iniciada: [x] Fase de supervisión iniciada: [ ] Gestionando el riesgo: [ ]
RESPONSABLE: Oljer Cando

Fuente: Autor

Tabla V.LII. Gestión del Riesgo 4

HOJA DE GESTIÓN DEL RIESGO			
ID DEL RIESGO: R4		**FECHA:**	
Probabilidad: Media **Valor: 2**	**Impacto:** **Alto** **Valor: 3**	**Exposición:** **Alta** **Valor: 6**	**Prioridad: 1**
DESCRIPCIÓN: costo del proyecto final exagerado			
REFINAMIENTO:			

Causas: No se planifico de manera adecuada el presupuesto para el desarrollo del sistema La culminación de cada etapa no fue a tiempo **Consecuencias:** Retraso en la planificación Aumento en el costo final del proyecto
REDUCCIÓN: Culminar las etapas asignadas puntualmente Realizar una correcta planificación en el presupuesto
SUPERVISIÓN: Revisar que la planificación presupuestaria sea adecuada para evitar inconvenientes Verificar el Trabajo para que al finalizar se tenga una aplicación eficiente
ESTADO ACTUAL: Fase de reducción iniciada: [x] Fase de supervisión iniciada: [ ] Gestionando el riesgo: [ ]
RESPONSABLE: Oljer Cando

Fuente: Autor

Tabla V.LIII. Gestión del Riesgo 5

<table>
<tr><td colspan="4">HOJA DE GESTIÓN DEL RIESGO</td></tr>
<tr><td colspan="2">ID DEL RIESGO: R5</td><td colspan="2">FECHA:</td></tr>
<tr><td>Probabilidad: Baja

Valor: 1</td><td>Impacto:

Bajo

Valor: 1</td><td>Exposición:

Baja

Valor: 2</td><td>Prioridad: 4</td></tr>
<tr><td colspan="4">DESCRIPCIÓN: Interfaces inadecuadas para validar las operaciones en el sistema</td></tr>
<tr><td colspan="4">REFINAMIENTO:

Causas:

El diseñador no elabore interfaces con datos necesarios para que el usuario valide la aplicación

Consecuencias:

Amenaza la calidad del software y la implementación puede llegar a ser difícil.</td></tr>
<tr><td colspan="4">REDUCCIÓN:

Diseñar correctamente las interfaces para validar el acceso al sistema de gestión</td></tr>
<tr><td colspan="4">SUPERVISIÓN:

Revisar durante el diseño de las interfaces que los datos sean los necesarios</td></tr>
<tr><td colspan="4">GESTIÓN:

Exigir que el las interfaces sean aprobadas por las partes involucradas</td></tr>
<tr><td colspan="4">ESTADO ACTUAL:

Fase de reducción iniciada: [x]

Fase de supervisión iniciada: [ ]</td></tr>
</table>

Gestionando el riesgo:
RESPONSABLE: Oljer Cando

Fuente: Autor

Tabla V.LIV. Gestión del Riesgo 6

HOJA DE GESTIÓN DEL RIESGO			
ID DEL RIESGO: R6		**FECHA:**	
Probabilidad: Baja **Valor:** 1	**Impacto:** moderado **Valor:** 2	**Exposición:** Baja **Valor:** 2	**Prioridad:** 3
DESCRIPCIÓN: Falta de comunicación con los miembros del municipio			
REFINAMIENTO: **Causas:** Ambiente de trabajo tenso No exista la confianza o celos por el nivel académico u ocupación. **Consecuencias:** Retraso en el proyecto.			
REDUCCIÓN: Fomentar la unión para el desarrollo de un proyecto que esté acorde a las expectativas			

SUPERVISIÓN: Actitud positiva de las personas involucradas en la actividad Ambiente amigable en el ambiente de trabajo y conseguir un proyecto eficiente
GESTIÓN: Los miembros entiendan la importancia del proyecto para que cooperen con lo que les corresponda El gestor de una charla clara con los involucrados en el proyecto.
ESTADO ACTUAL: Fase de reducción iniciada: [x] Fase de supervisión iniciada: [ ] Gestionando el riesgo: [ ]
RESPONSABLE: Oljer Cando

Fuente: Autor

Tabla V.LV. Gestión del Riesgo 7

HOJA DE GESTIÓN DEL RIESGO			
ID DEL RIESGO: R7		**FECHA:**	
Probabilidad: Media **Valor:** 1	**Impacto: A**lta **Valor:** 2	**Exposición:** Alta **Valor:** 2	**Prioridad:** 1
DESCRIPCIÓN: La información que se modifique no se actualice correctamente en			

la base de datos, del sistema
REFINAMIENTO: **Causas:** No se cuente con un método adecuado para facilitar la actualización automática **Consecuencias:** Inconsistencia en los datos
REDUCCIÓN: Validar los datos continuamente con los cosos posibles
SUPERVISIÓN: Revisar que los métodos estén de acuerdo a los modelos de las bases de datos y sean correcto
GESTIÓN: Revisar que el método cumpla la función con la que fue creado
ESTADO ACTUAL: Fase de reducción iniciada: [x] Fase de supervisión iniciada: [ ] Gestionando el riesgo: [ ]
RESPONSABLE: Oljer Cando

Fuente: Autor

5.7 PLANIFICACIÓN INICIAL

5.7.1 FACTIBILIDAD

Factibilidad Técnica

La factibilidad técnica ayuda a determinar si la propuesta puede ser implementada con el hardware, software y recurso humano disponible.

Para el desarrollo de la aplicación web “Sistema de gestión de Bienes” se cuenta con casi todos los recursos hardware y software necesarios. A continuación se detalla el hardware, software existente, requerido así como también el personal técnico requerido para el desarrollo del mismo.

Hardware Existente

Hardware con el que se cuenta para el desarrollo de la aplicación es el siguiente:

Tabla V.LVI. Hardware existente

Cantidad	Descripción	Observación
1	Computadora	Desarrollo de la aplicación y documentación
1	Infraestructura de Red	Acceder al internet para consultar las dudas en el desarrollo de la aplicación y realizar las pruebas respectivas

Fuente: Autor

Hardware Requerido

Tabla V.LVII. Hardware requerido

Cantidad	Descripción	Observación
1	Impresora	Imprimir informes
1	Servidor	Para pruebas de configuración del servidor

Fuente: Autor

Software Existente

Tabla V.LVIII. Software existente

Nombre	Descripción
Windows 7 Profesional	Sistema Operativo
Microsoft Visio 2013	Herramienta para diseño UML

Fuente: Autor

Software Requerido

Tabla V.LIX. Software requerido

Nombre	Descripción
NetBeans IDE 7.2.1	Entorno de desarrollo
Ireport 3.7.0	Reportero
Glassfish 3.0	Servidor de aplicaciones Web
Netsparker Community	Analiza Vulnerabilidades existentes en un sistema web
Practiline Source	Cuenta las líneas de código

Fuente: Autor

Recurso Humano Requerido

Tabla V.LX. Recurso Humano Requerido

Función	Formación
Jefe de Proyecto	Ingeniería en Sistemas
Equipo de desarrolladores	Estudiante de Ingeniería en Sistemas
Administrador de Base de Datos	Ingeniería en Sistemas

Diseñador	Estudiante de Ingeniería en Sistemas
Jefe de Proyecto	Ingeniería en Sistemas

Fuente: Autor

Factibilidad Operativa

Recurso Humano

El recurso humano que participa en la operación del sistema son:

- Usuarios Directos

Los usuarios directos a capacitar para el manejo del sistema son:

Personal a Capacitar

Tabla V.LXI. Personal a Capacitar

Nombre	**Función**
U. Superadministrador	Jefe responsable de la Unidad de Sistemas
U. Administrador	Directores de los Departamentos
U. Normales	Alcalde

Fuente: Autor

Factibilidad Económica

El tiempo de duración de este proyecto será de 12 meses

Costos

Costos de desarrollo

Costos Personal

		Mensual Total
Jefe de Proyecto y Desarrollador	$800	$9.600,00
Administrador de BD, Diseñador	$800	$3.200,00
Costo Personal Total		**$12.800,00**

Costo de hardware y software

Costo Software	
Internet	**$400,00**
Costos varios	
Suministros	$300.00
Alimentación	$800.00
Papel A4	$10.00
Costo Varios Total	**$1.210,00**
Costos Total de desarrollo	**$14.410,00**

Análisis Costo Beneficio

Los beneficios que se podrá obtener con la utilización de este sistema son los siguientes:

Permitirá realizar el proceso de gestión de bienes (materiales) existentes para la obras de una forma eficiente y real, permitiendo organizar, reducir el flujo de trabajo de los departamentos, además usar la tecnología existen en la institución y reducir la contaminación con el uso de papel, tinta, cartuchos y control del uso de los materiales destinado a los lugares para los que fueron adquiridos.

Se podrá realizar un mayor control del uso de los combustibles y de los bienes que existen en la institución.

Mediante el sistema se puede generar reportes que ayudarán a la toma de decisiones de las autoridades.

Se optimizará el tiempo en la elaboración de informes.

La utilización del sistema será fácil de utilizar y además los usuarios pueden ingresar desde cualquier lugar.

7.8 FASE DE PLANIFICACIÓN

Obtener un cronograma de trabajo que cumpla con lo especificado en la fase de visión, desarrollar los requerimientos funcionales y no funcionales, describir el escenario, diagramar casos de uso

PLANEACIÓN

En esta fase se realiza la preparación de la especificación funcional, diseño conceptual.

Especificación Funcional

5.8.1 DISEÑO CONCEPTUAL

5.8.1.1.Requerimientos Funcionales

MODULO ADMINISTRACION DE USUARIOS

5.8.1.1.1. RequerimientoFuncional 1

5.8.1.1.1.1.Especificaciones

5.8.1.1.1.1.1.Introducción

El sistema deberá permitir gestionar datos de usuarios

5.8.1.1.1.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.1.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero
4. Ingresara al sistema
5. Caso Contrario
6. Mensajes de error
7. Ingreso
8. Aceptar
9. Validación de datos
10. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.1.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.1.2.Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.1.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.1.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO DEPARTAMENTO

5.8.1.1.2 Requerimiento Funcional 2

5.8.1.1.2.1. Especificaciones

5.8.1.1.2.1.1. Introducción

El sistema deberá permitir gestionar datos del Departamento

5.8.1.1.2.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.2.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero
4. Ingresara al sistema

5. Caso Contrario
6. Mensajes de error
7. Ingreso
8. Aceptar
9. Validación de datos
10. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.2.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.2.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.2.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.2.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO REPRESENTANTES

5.8.1.1.3. RequerimientoFuncional 3

5.8.1.1.3.1. Especificaciones

5.8.1.1.3.1.1. Introducción

El sistema deberá permitir asignar datos de los representantes.

5.8.1.1.3.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.3.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.3.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.3.1.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.3.1.2. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.3.1.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO DE ASIGNACION DE RECURSOS

5.8.1.1.4. RequerimientoFuncional 4

5.8.1.1.4.1. Especificaciones

5.8.1.1.4.1.1. Introducción

El sistema deberá permitir asignar datos de los recursos.

5.8.1.1.4.1.4. Salidas

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.4.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.4.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.4.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.4.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.

- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.4.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO ADMINISTRACION DE COMPUTADORAS

5.8.1.1.5. RequerimientoFuncional 5

5.8.1.1.5.1. Especificaciones

5.8.1.1.5.1.1. Introducción

El sistema deberá permitir asignar datos de los mantenimientos

El sistema deberá permitir asignar datos de las partes

5.8.1.1.5.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.5.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.5.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.5.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.5.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.5.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO COMPRAS BODEGA

5.8.1.1.6. RequerimientoFuncional 6,7,8

5.8.1.1.6.1. Especificaciones

5.8.1.1.6.1.1. Introducción

El sistema deberá permitir asignar datos de las órdenes de compra.

El sistema deberá permitir asignar datos de las órdenes de entrega.

5.8.1.1.6.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.6.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.6.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.6.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.6.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.6.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO SOLICITUD DE ENTREGA BODEGA

5.8.1.1.7. RequerimientoFuncional 9,10

5.8.1.1.7.1. Especificaciones

5.8.1.1.7.1.1. Introducción

El sistema deberá permitir asignar datos de las solicitudes de entrega.

El sistema deberá permitir asignar datos de los ítems de las órdenes de entrega.

5.8.1.1.7.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.7.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.7.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.7.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.7.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.7.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO PROVEEDORES

5.8.1.1.8. RequerimientoFuncional 11

5.8.1.1.8.1. Especificaciones

5.8.1.1.8.1.1. Introducción

El sistema deberá permitir asignar datos de los proveedores.

5.8.1.1.8.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.8.1.3. Procesos

8. El usuario deberá ingresara el usuario y contraseña de autenticación.
9. El sistema validad el usuario y contraseña
10. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

11. Ingreso

12. Aceptar

13. Validación de datos

14. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.8.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.8.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.8.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.8.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO CATEGORIA

5.8.1.1.9. RequerimientoFuncional 12

5.8.1.1.9.1. Especificaciones

5.8.1.1.9.1.1. Introducción

El sistema deberá permitir asignar datos de las categorías.

5.8.1.1.9.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.9.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.9.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.9.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.9.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.9.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO UNIDAD MEDIDA

5.8.1.1.10. RequerimientoFuncional 13

5.8.1.1.10.1. Especificaciones

5.8.1.1.10.1.1. Introducción

El sistema deberá permitir asignar datos de las unidades de medida.

5.8.1.1.10.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.10.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.10.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.10.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.10.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.

- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.10.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO PRODUCTOS

5.8.1.1.11. RequerimientoFuncional 14

5.8.1.1.11.1. Especificaciones

5.8.1.1.11.1.1. Introducción

El sistema deberá permitir asignar datos de los productos.

5.8.1.1.11.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.11.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso

5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.11.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.11.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.11.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.11.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO SOLICITUD DE COMPRAS DEPARTAMENTOS

5.8.1.1.12. RequerimientoFuncional 15

5.8.1.1.12.1. Especificaciones

5.8.1.1.12.1.1. Introducción

El sistema deberá permitir asignar datos de las solicitudes de compra a Bodega.

5.8.1.1.12.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.12.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.12.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.12.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.12.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.12.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO SOLICITUD ENTREGA DEPARTAMENTOS

5.8.1.1.13. RequerimientoFuncional 16

5.8.1.1.13.1. Especificaciones

5.8.1.1.13.1.1. Introducción

El sistema deberá permitir asignar datos de las solicitudes de entrega a Bodega.

5.8.1.1.13.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.13.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.13.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.13.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.13.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.

- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.13.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO LUGARES

5.8.1.1.14. RequerimientoFuncional 17

5.8.1.1.14.1. Especificaciones

5.8.1.1.14.1.1. Introducción

El sistema deberá permitir asignar datos de los lugares.

5.8.1.1.14.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.14.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.14.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.14.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.14.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.14.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO OBRAS

5.8.1.1.15. RequerimientoFuncional 18

5.8.1.1.15.1. Especificaciones

5.8.1.1.15.1.1. Introducción

El sistema deberá permitir asignar datos de las obras

5.8.1.1.15.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.15.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.15.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.15.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.15.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.15.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO VEHICULOS

5.8.1.1.16. RequerimientoFuncional 19

5.8.1.1.16.1. Especificaciones

5.8.1.1.16.1.1. Introducción

El sistema deberá permitir asignar datos de los Vehículos.

5.8.1.1.16.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.16.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.16.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.16.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.16.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.

- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.16.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO ENTREGA COMBUSTIBLE

5.8.1.1.17. RequerimientoFuncional 20

5.8.1.1.17.1. Especificaciones

5.8.1.1.17.1.1. Introducción

El sistema deberá permitir asignar datos de las órdenes de entrega.

5.8.1.1.17.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.17.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso

5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.17.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.17.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.17.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.17.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

MODULO SOLICITUD DE ENTREGA DEPARTAMENTOS

5.8.1.1.18. RequerimientoFuncional 21

5.8.1.1.18.1. Especificaciones

5.8.1.1.18.1.1. Introducción

El sistema deberá permitir asignar datos de las solicitudes de entrega.

5.8.1.1.18.1.2. Entrada

Fuente de Entrada

Usuario

Contraseña

Frecuencia

Bajo demanda

Requisitos de Control

Controla que los campos del formulario no estén vacíos y que los datos sean los correctos

Entradas validas

Todos los campos sean válidos.

5.8.1.1.18.1.3. Procesos

1. El usuario deberá ingresara el usuario y contraseña de autenticación.
2. El sistema validad el usuario y contraseña
3. Si el dato es verdadero

a. Ingresara al sistema

Caso Contrario

b. Mensajes de error

4. Ingreso
5. Aceptar
6. Validación de datos
7. Si todos los campos están llenos y los datos son correctos

a. Actualización de la base de datos y visualización de resultados

Caso Contrario

b. Mensaje de error

5.8.1.1.18.1.4. Salidas

Destino de las salidas

Muestra un listado nuevo con el dato ingresado cuando los datos ingresados son correctos o sin el dato elegido.

5.8.1.1.18.2. Interfaces de hardware

- El monitor es el principal medio hardware de visualización, el cual se lo utilizará para mostrar cada uno de los procesos que se efectuarán.

5.8.1.1.18.3. Interfaces de Software

- La herramienta de desarrollo que se utilizara es NetBeans IDE 7.2.1 para realizar las aplicaciones que cumplan el requerimiento.
- PostgreSQL 9.2 será el repositorio de datos.

5.8.1.1.18.4. Interfaces de Software

El sistema utilizara el protocolo TCP/IP para establecer la comunicación.

5.8.1.1 Requerimientos no Funcionales

Requerimientos no funcionales con sus perspectivas

5.8.1.1.1 Rendimiento

Tiempos de respuesta para acceder a la página de autenticación del sistema máximo de 30 segundos.

5.8.1.1.2 Disponibilidad

El sistema estará fuera de línea máximo 30 minutos

Empleo de sistemas de respaldo

5.8.1.1.3 Seguridad

Para ingresar al sistema primero deberá el usuario autenticarse, y según el tipo asignado por el administrador accederá al ambiente de trabajo correspondiente.

5.8.1.1.4 Portabilidad

Garantizar compatibilidad con los otros sistemas operativos: Windows xp, Windows 7, Linux

5.8.1.1.5 Mantenibilidad

Documentación del diseño y la codificación de la solución

5.8.1.1.6 Escalabilidad

Diseño de la arquitectura empleando módulos independientes

5.8.1.1.7 Interfaces

Interfaces realizadas en NetBeans 7.2.1

5.8.1.1.8 Usabilidad

Factibilidad de uso

5.8.1.2 Actores

SuperAdministrador

Es la persona que posee el control total de todo el Sistema, y es el encargado de otorgar los diferentes permisos a los demás Usuarios además él se encargará de asignar las personas responsables de evaluar cada carrera.

Administrador Normal

Serán los encargados de administrar en su totalidad el sistema desarrollado pero solo de las carreras que le ha sido asignada.

- Control Total
- Lectura.
- Escritura.
- Gestión de reportes
    - Generar.
    - Imprimir.
    - Visualizar

Usuario Invitado

Es un tipo de usuario limitado

- Solo lectura
- Visualizar reportes de forma general

5.8.1.2.1 Casos de Uso

CU - SUPER ADMINISTRADOR

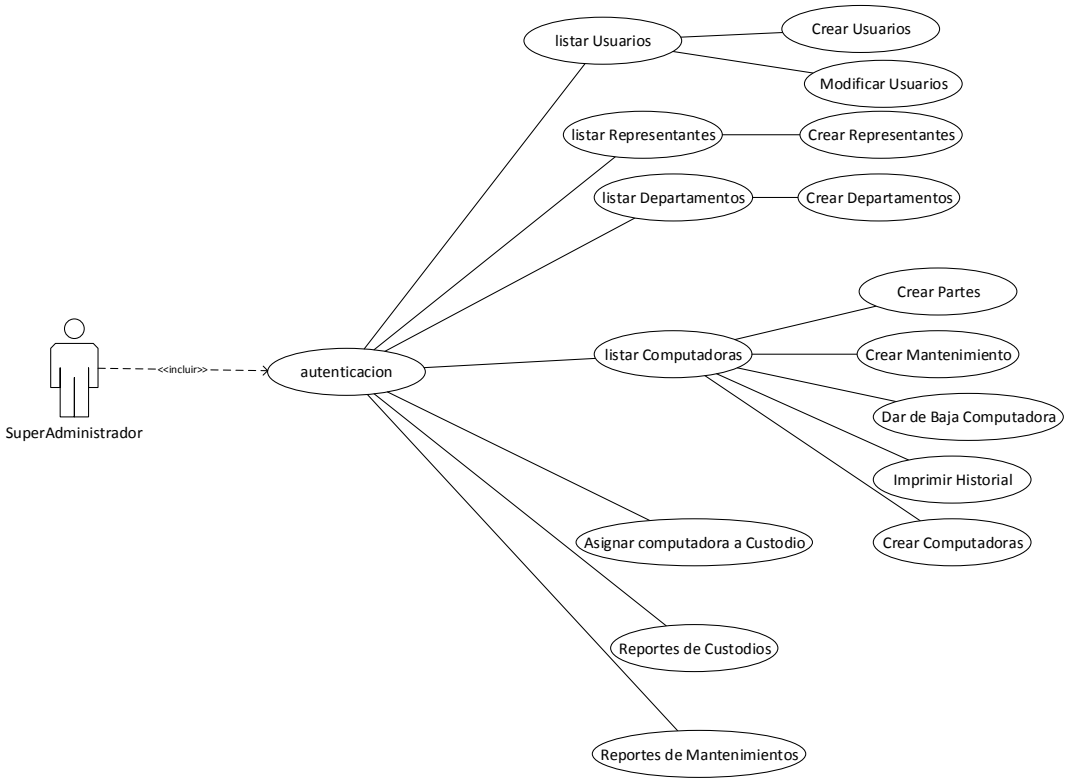


Figura V.31. Caso de uso Súper Administrador

Fuente: Autor

Los siguientes casos de uso se les agrupado dentro de la categoría administrado pero los mismo se desarrollan en diferentes ambientes.

CU-ADMINISTRADOR DEPARTAMENTO

El usuario Administrador Departamento tendrá control sobre todo el ambiente de trabajo asignado.

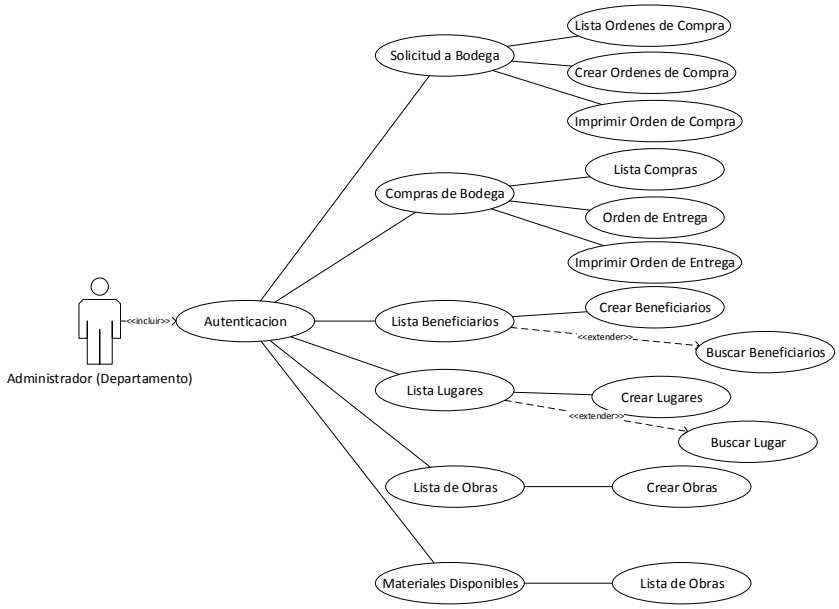


Figura V.32. CU Administrador Departamento

Fuente: Autor

CU ADMINISTRADOR BODEGA

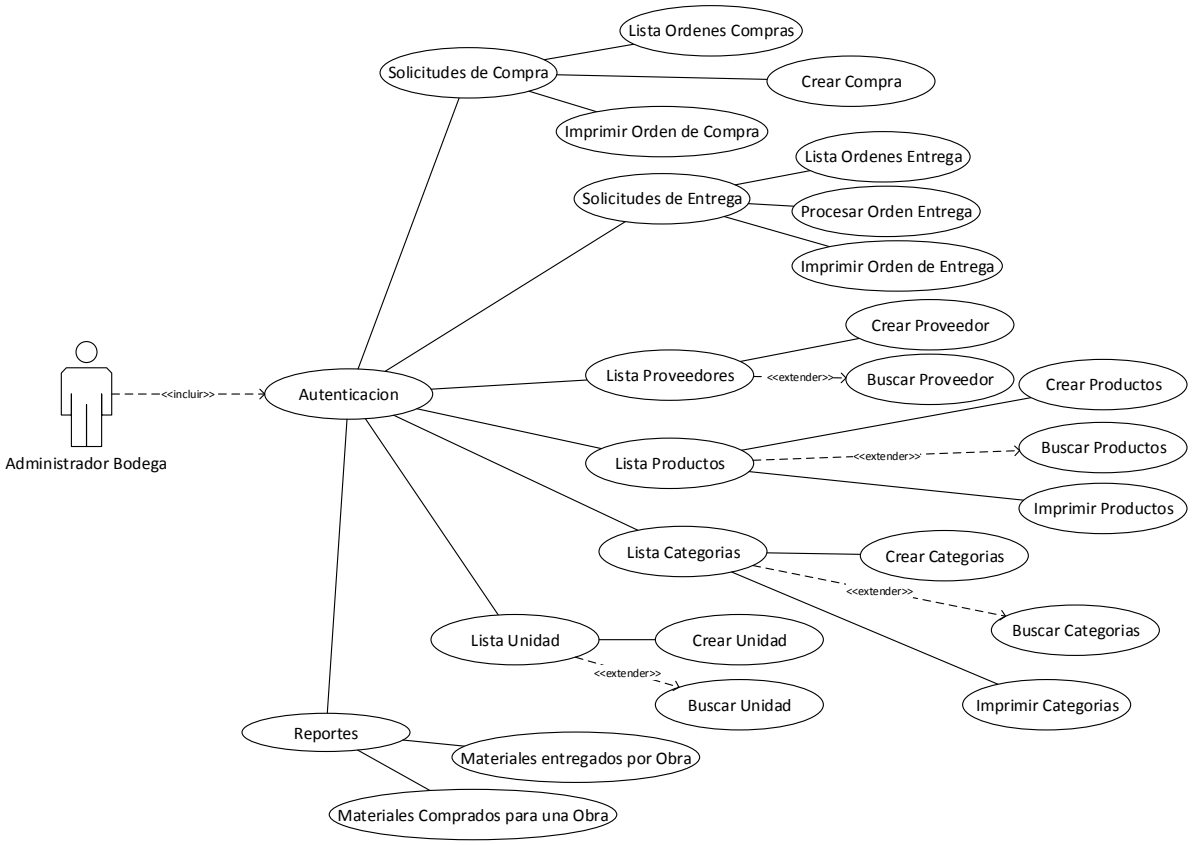


Figura V.33.Caso Uso Administrador (bodega)

Fuente: Autor

CU – ADMINISTRADOR (COMBUSTIBLE)

Figura V.34.Caso de Uso Administrador (Combustible)

Fuente: Autor

CU – ADMINISTRADOR MANTENIMIENTO

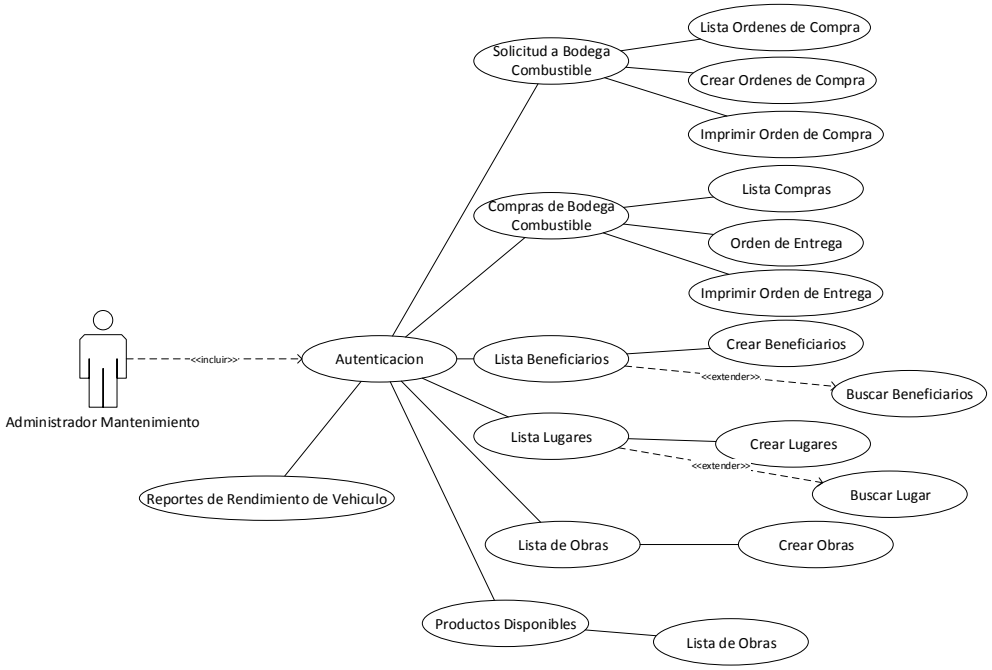


Figura V.35.Casos de Uso Administrador (Mantenimiento)

Fuente: Autor

CU – INVITADO

Solo tienen acceso a reportes generales que a continuación se detalla.

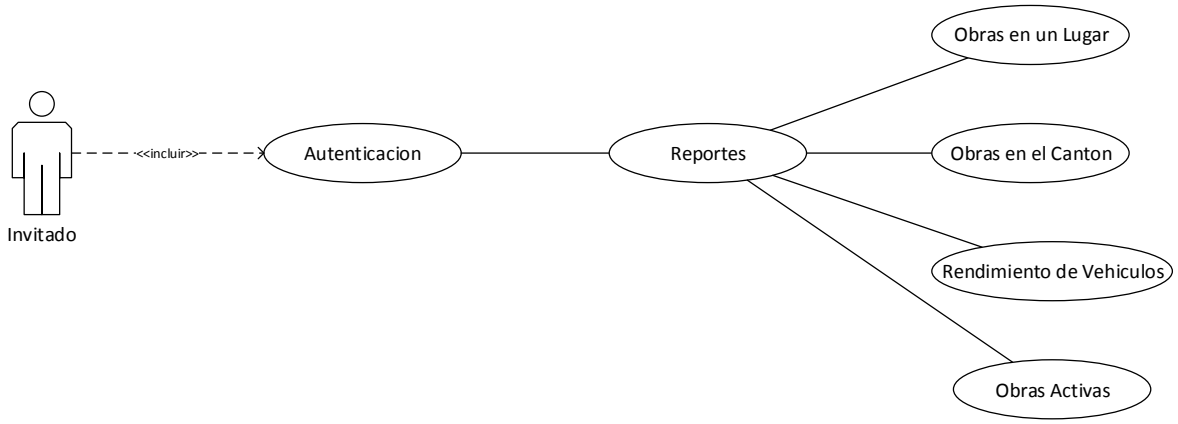


Figura V.36.Casos de Uso Invitado

Fuente: Autor

5.8.1.2.2 Escenarios

AUTENTICACION USARIO SUPERADMINISTRADOR

- **Personas Alternativas**

Tabla V.LXII. Personas Alternativas para Super Administrador

PERSONA	DESVIACIONES(Si es aplicable)
Persona delegada por el Responsable de la unidad de Sistemas del GAD Municipal de Arosemena Tola	Ninguna

Fuente: Autor

- **Descripción del Escenario**

El **SuperAdministrador** ingresará su Usuario, Clave en la página de autenticación. Este administrador creara usuarios y el ambiente de trabajo como departamentos, representantes.

- **Personas Alternativas**

Tabla V.LXIII. Personas alternativas para Administradores

PERSONA	DESCIACIONES
Responsables de los departamentos que genera obras o que esté involucrado en la obra	Ninguna
Mantenimiento	Ninguna
Bodega	Ninguna
Combustible	Ninguna

Fuente: Autor

- **Descripción del Escenario**

El Administrador Ingresara su Usuario y Contraseña en la página de autenticación. Si los datos son los correctos se presentara la página principal

AUTENTICACION USUARIOS INVITADOS

Tabla V.LXIV. Personas alternativas para usuarios tipo invitado

PERSONA	DESVIACIONES(Si es aplicable)
Alcalde	Ninguna
Persona elegida por las máximas autoridades del GAD Municipal de Arosemena Tola	Ninguna

Fuente: Autor

- **Descripción del Escenario**

Los usuarios de tipo Invitado también tienen un usuario y clave que les asigna el SuperAdministrador, estos datos son validados en la base de datos de PostgreSQL 9.2.

5.8.1.2.3 Glosario de Términos

Tabla V.LXV. Glosario de Términos

TERMINO	DESCRIPCION
MSF	Microsoft Solution Framework, metodología empleada para el desarrollo de la aplicación.
UML	Lenguaje Unificado de Modelado, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.
Módulos	Distintas dependencias del sistema que brinda ciertos servicios a los usuarios.
Hardware	Las características físicas y tangibles del computador utilizado.
Clave	Es una serie secreta de caracteres que permite a un usuario tener acceso a un ordenador, programa, sitio.
Rendimiento	Capacidad de evolución y aprovechamiento del sistema.
Servidor	Un servidor es un ordenador de gran potencia, que se encarga de "prestar un servicio" a otros ordenadores que se conectan a el
Sistema Operativo	Es un programa informático que actúa de interfaz entre los dispositivos de hardware y el usuario. Es responsable de gestionar, coordinar las actividades y llevar a cabo el intercambio de recursos de un computador.
Base de Datos	Conjunto de datos organizados para su almacenamiento en la memoria de un ordenador o computadora, diseñado para facilitar su mantenimiento y acceso de una forma estándar. La información se organiza en campos y registros.
Usuario	Persona que utiliza el sistema
Administrador	Es la persona encargada de administrar en su totalidad el sistema, además puede generar reportes.

Reporte	El reporte es aquel documento que se utilizará cuando se quiera obtener o dar información de una determinada cuestión.
Metodología	Conjunto de pasos lógicos y ordenados a seguir para realizar una actividad, la metodología son micro actividades. Primero debemos tener un modelo para seguir una metodología.
Internet	Conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.
Evaluación	Es la determinación sistemática del mérito, el valor y el significado de algo o alguien en función de unos criterios respecto a un conjunto de normas.
Metodología	Conjunto de pasos lógicos y ordenados a seguir para realizar una actividad, la metodología son micro actividades. Primero debemos tener un modelo para seguir una metodología.

Fuente: Autor

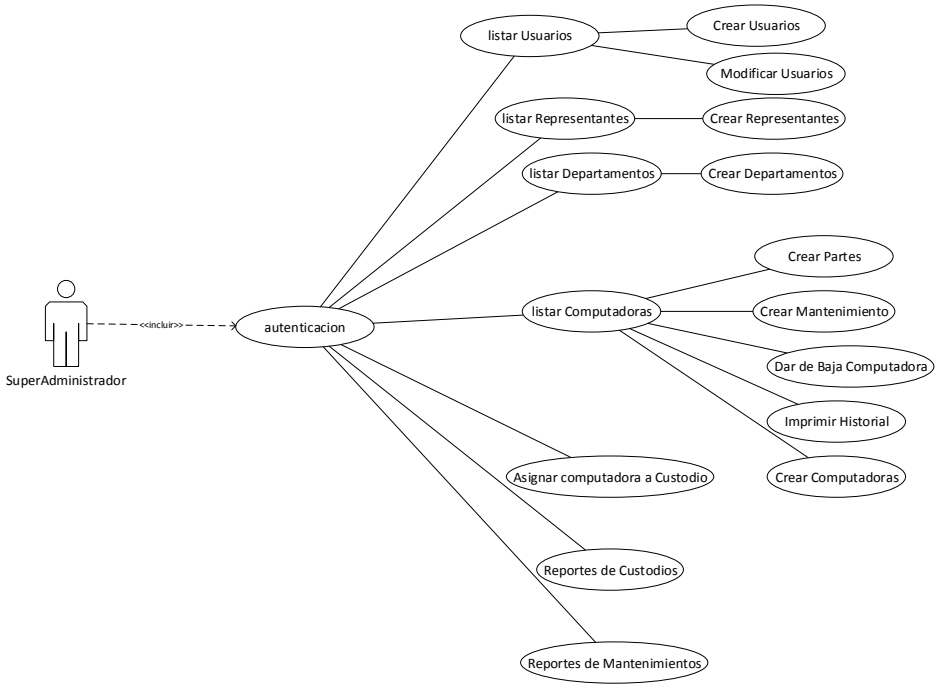
5.8.1.2.4 Refinar los Casos de Uso

Tabla V.LXVI. Autenticación de usuarios

IDENTIFICAR CASO DE USO	CU - SeperAdministrador
NOMBRE DEL CASO DE USO	Realizar administración del sistema
ACTORES	Responsable de la Unidad de Sistemas
PROPOSITO	Crear el ambiente correcto de trabajo del sistema
VISION GENERAL	El responsable para realizar los

	procesos correspondientes deberá autenticarse previamente usando su usuario y contraseña.
TIPO	Primario, real y extendido
REFERENCIAS	Caso de Uso: SuperAdministrador
CURSO TIPICO DE EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DEL SISTEMA**
1. Este caso de uso empieza cuando el Administrador desea crear el ambiente de trabajo para el sistema.	2. El sistema cuenta con un control para que se autentique.
3. Autenticación, ingresa usuario y contraseña	4.Valida los datos ingresados
	5. Al autenticarse correctamente podrá llevar a cabo los procesos correspondientes.
CURSOS ALTERNATIVOS	
2.1.- El sistema no presenta pantallas para el ingreso de datos.	
Control de Campos Vacíos 4.1.- Ingreso de Usuario Cuando no se ingresa algún campo si es obligatorio le informa del error solicitando que sea lleno. 4.2.- Si el usuario no está en la base de datos, el sistema le informara que no existe, que se acerque al administrador.	

Fuente: Autor



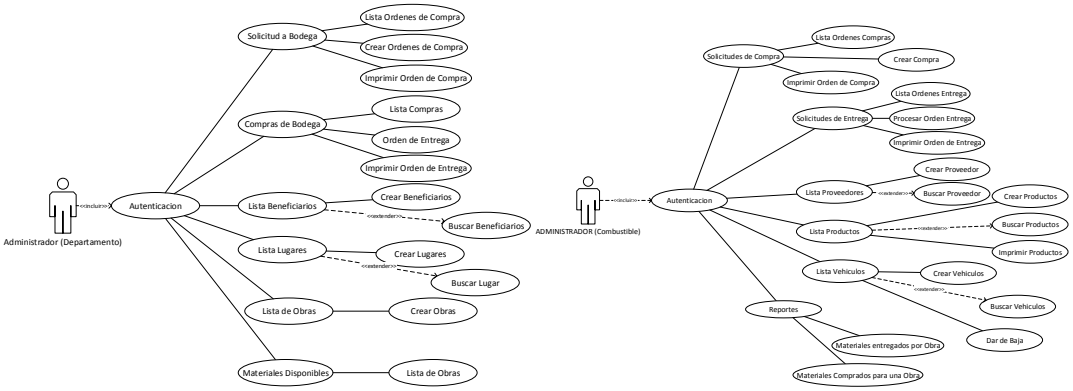
CU – USUARIO ADMINISTRADOR (Departamento, Bodega, Mantenimiento, Combustible)

Tabla V.LXVII. Usuario Administrador

IDENTIFICAR CASO DE USO	CU - Administrador
NOMBRE DEL CASO DE USO	Realizar administración del sistema
ACTORES	Responsable de los departamentos involucrados en una Obra
PROPOSITO	Gestionar y controlar el correcto uso de los bienes
VISION GENERAL	El responsable para realizar los procesos correspondientes deberá autenticarse previamente usando su usuario y contraseña.
TIPO	Primario, real y extendido
REFERENCIAS	Caso de Uso: Administrador

	Departamento, Administrador Bodega, Administrador Mantenimiento, Administrador Combustible
CURSO TIPICO DE EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DEL SISTEMA**
1. Este caso de uso empieza cuando el Administrador desea crear el ambiente de trabajo para el sistema.	2. El sistema cuenta con un control para que se autentique.
3. Autenticación, ingresa usuario y contraseña	4.Valida los datos ingresados
	5. Al autenticarse correctamente podrá llevar a cabo los procesos correspondientes.
	6. Presenta la página con el menú de opciones correspondientes
CURSOS ALTERNATIVOS	
2.1.- El sistema no presenta pantallas para el ingreso de datos.	
Control de Campos Vacíos 4.1.- Ingreso de Usuario Cuando no se ingresa algún campo si es obligatorio le informa del error solicitando que sea lleno. 4.2.- Si el usuario no está en la base de datos, el sistema le informara que no existe, que se acerque al administrador.	

Fuente: Autor



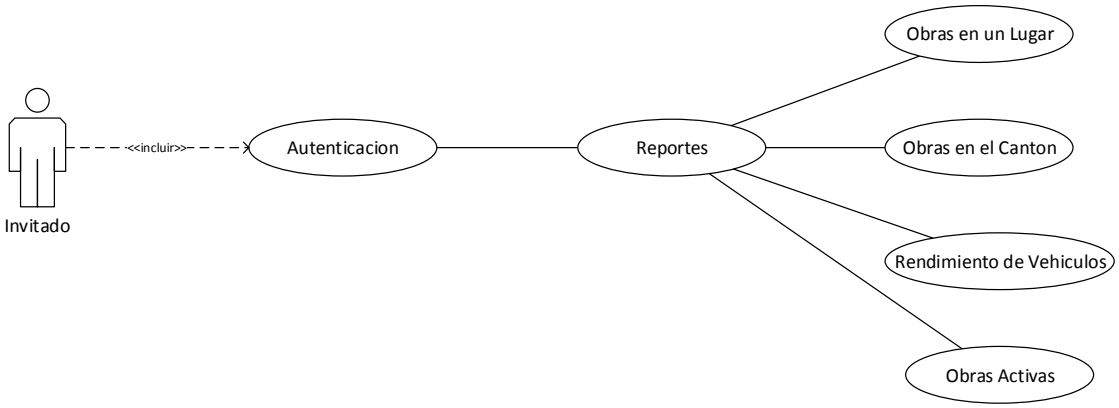
CU – USUARIO INVITADO

Tabla V.LXVIII. Usuario invitado

IDENTIFICAR CASO DE USO	CU - Administrador
NOMBRE DEL CASO DE USO	Realizar Consultas del Funcionamiento
ACTORES	ALCALDE
PROPOSITO	Realizar el seguimiento del uso de combustible, consultar obras en un sector
VISION GENERAL	El responsable para realizar los procesos correspondientes deberá autenticarse previamente usando su usuario y contraseña, luego ingresara al menú de opciones correspondientes.
TIPO	Primario, real y extendido
REFERENCIAS	Caso de Uso: Administrador Departamento, Administrador Bodega, Administrador Mantenimiento, Administrador

	Combustible
CURSO TIPICO DE EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DEL SISTEMA**
1. Este caso de uso empieza cuando el Administrador desea crear el ambiente de trabajo para el sistema.	2. El sistema cuenta con un control para que se autentique.
3. Autenticación, ingresa usuario y contraseña	4.Valida los datos ingresados
	5. Al autenticarse correctamente podrá llevar a cabo los procesos correspondientes.
	6. Presenta la página con el menú de opciones correspondientes
CURSOS ALTERNATIVOS	
2.1.- El sistema no presenta pantallas para el ingreso de datos.	
Control de Campos Vacíos 4.1.- Ingreso de Usuario Cuando no se ingresa algún campo si es obligatorio le informa del error solicitando que sea lleno. 4.2.- Si el usuario no está en la base de datos, el sistema le informara que no existe, que se acerque al administrador.	

Fuente: Autor



5.8.2 Diseño Lógico

5.8.2.1 Tecnología a utilizar en el proyecto

- Entorno de Desarrollo Java NetBeans IDE 7.2.1
- Base de Datos PostgreSQL 9.2
- Documentación MSF

5.8.2.2 Diagrama de Secuencia

AUTENTICACIÓN DE USUARIOS

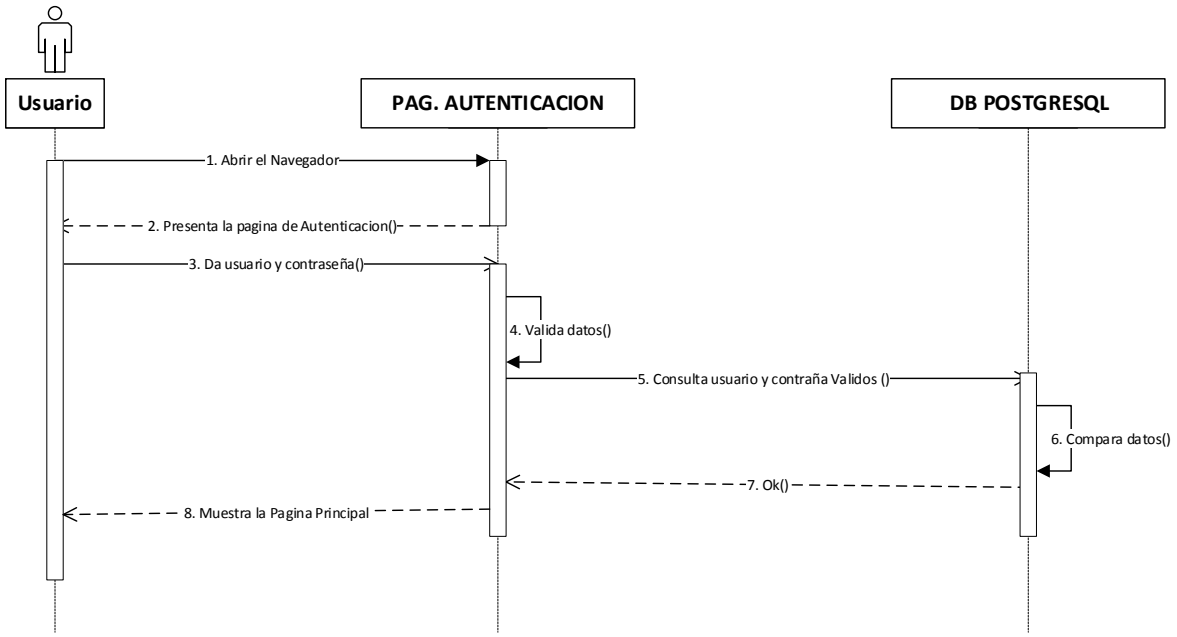


Figura V.37. Diagrama de Secuencia de Autenticación de usuarios

Fuente: Autor

Los siguientes Diagramas de secuencias son generales para todos los módulos existentes, en este caso realiza para crear un Departamento

CREAR DEPARTAMENTO

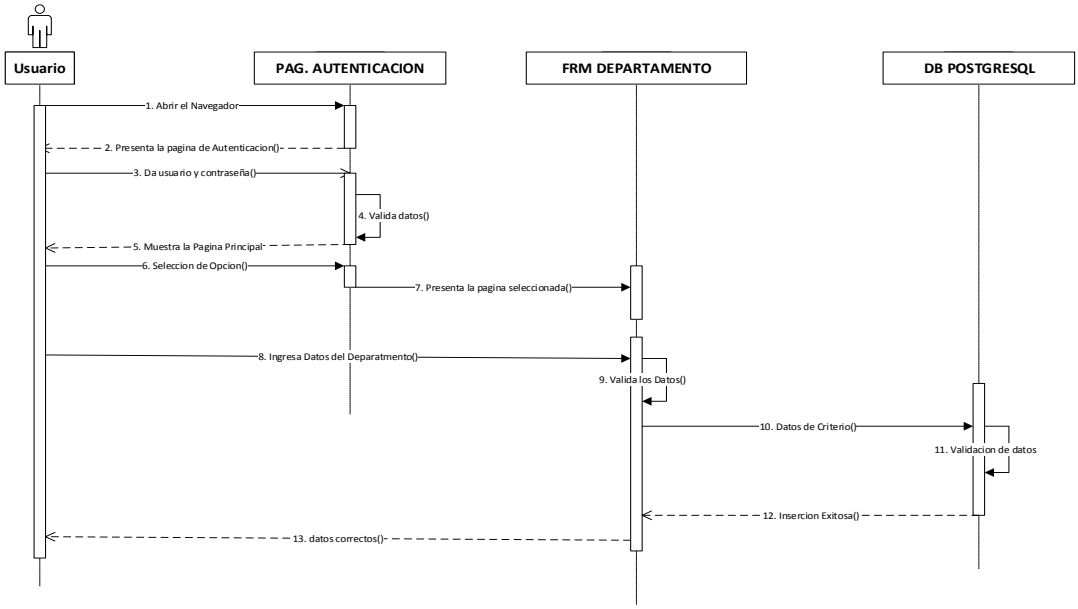


Figura V.38. Diagrama de secuencias para la creación de un Departamento

Fuente: Autor

DAR DE BAJA UNA COMPUTADORA

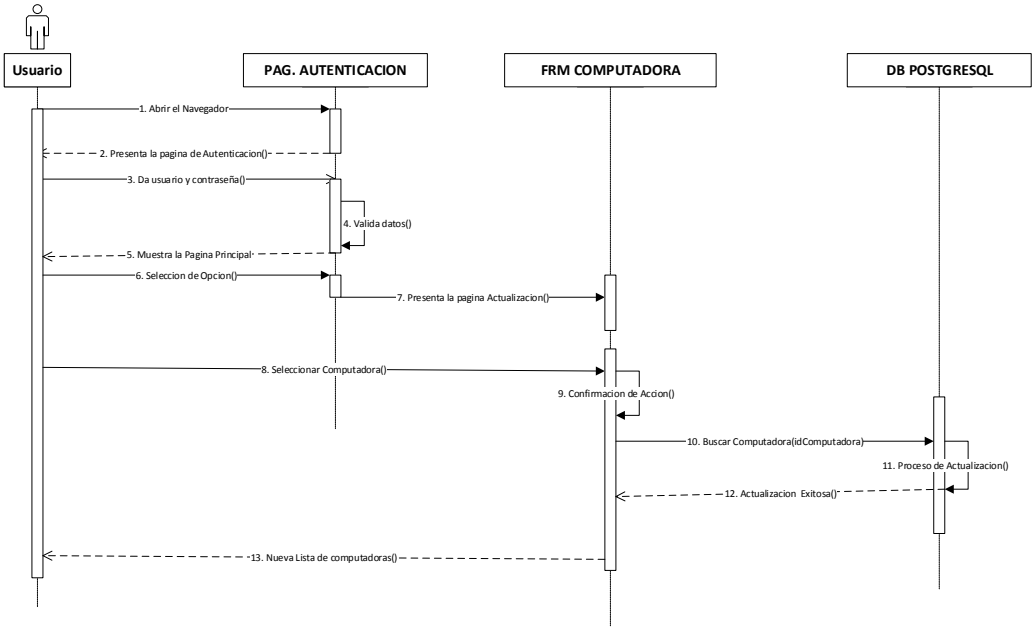


Figura V.39.Diagrama de Secuencia para dar de baja un equipo

Fuente: Autor

5.8.2.3 Diagrama de Clase

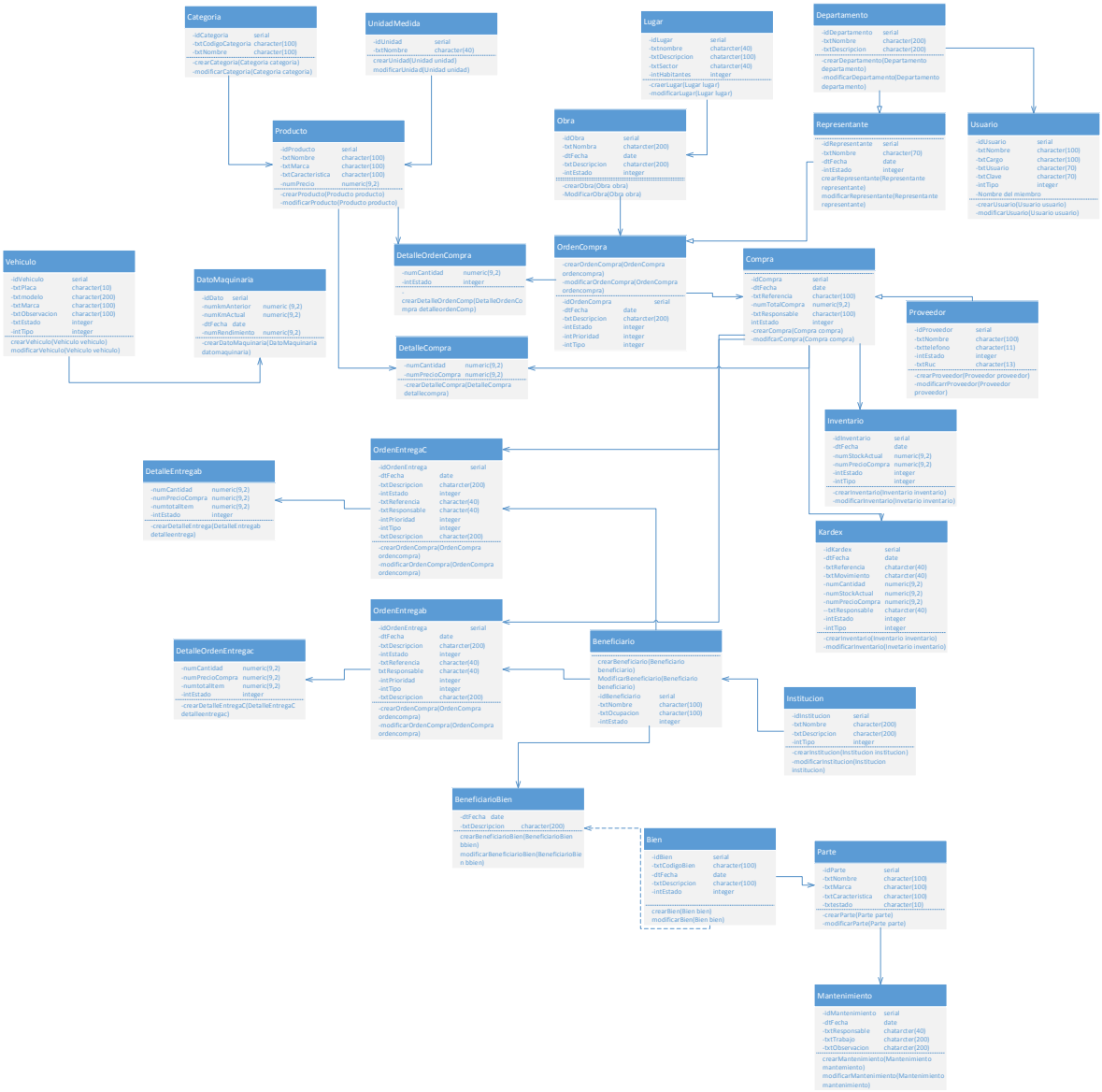


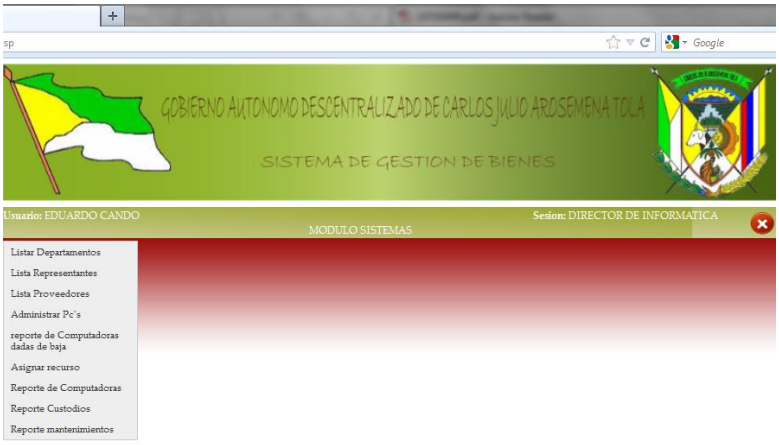
Figura V.40. Diagrama de Clase

Fuente: Autor

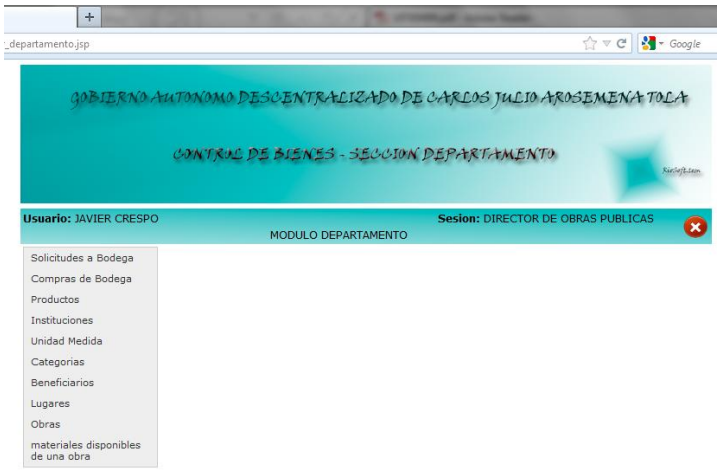
5.8.2.4 Diseño de Interfaces

- Autenticación de Usuarios

- Página Principal del SuperAdministrador



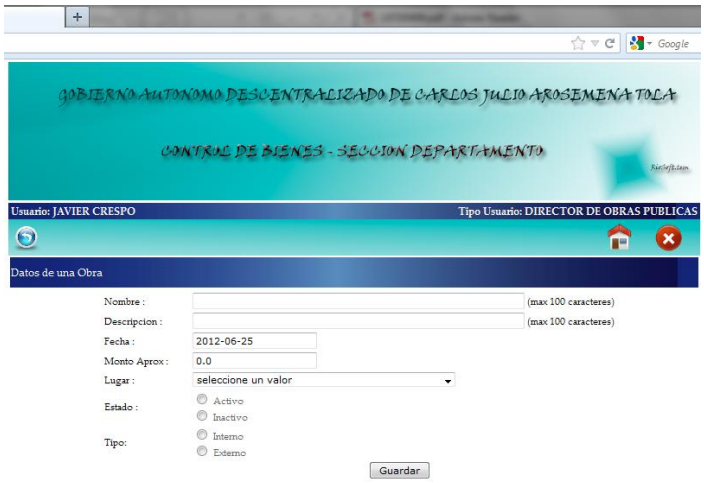
- Gestión de Bienes Departamento



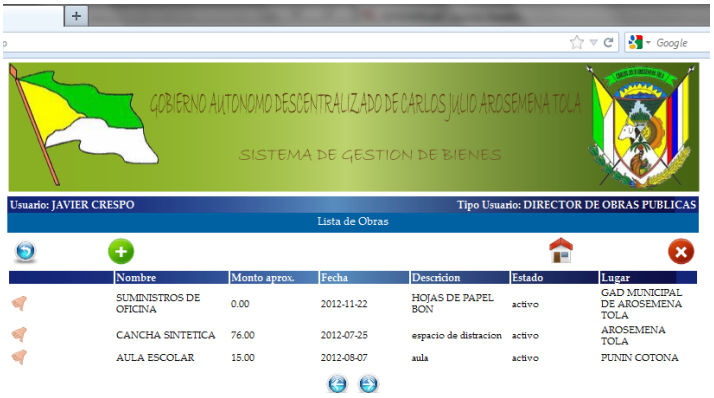
- Lista de Productos

- Lista de Órdenes de Compra

- Creación de Una obra por el Departamento de obras Publicas



- Dar de Baja una Obra



5.8.3 Diseño Físico

5.8.3.1 Diagrama de Actividades

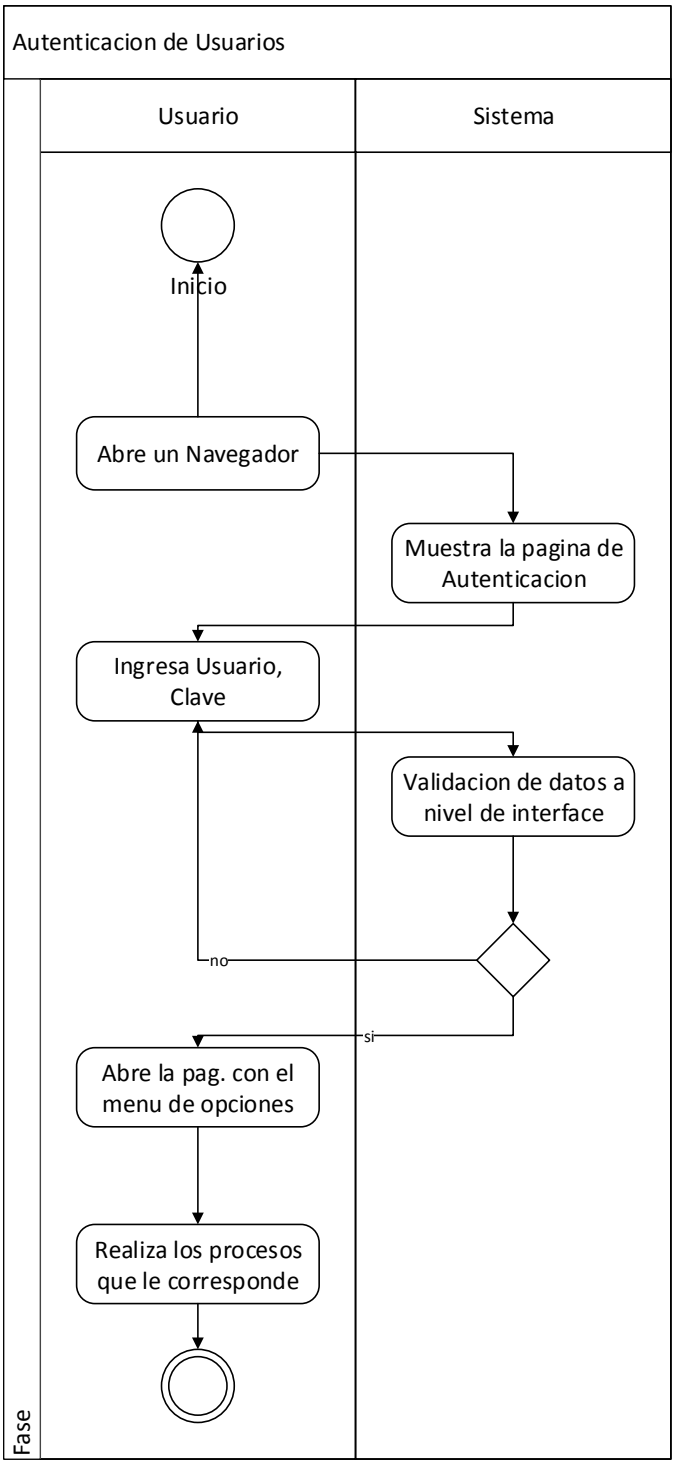


Figura V. 2. Diagrama de Actividades

Fuente: Autor

5.8.3.2 Diagrama de Componentes

5.8.3.3 Diagrama de Implementación

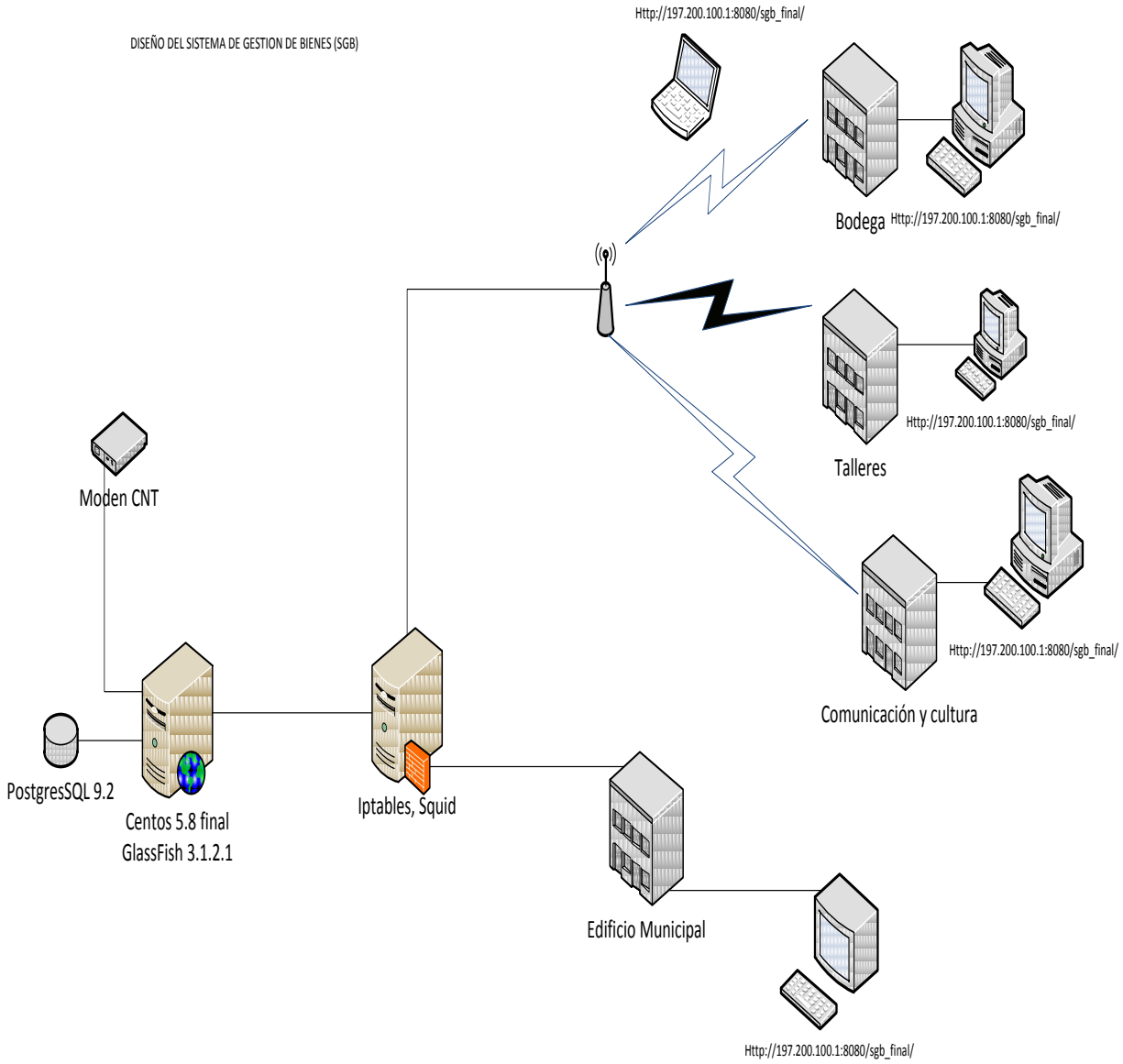


Figura V. 3. Diagrama de Componentes

Fuente: Autor

5.8.3.4 Modelo Físico de la Base de Datos

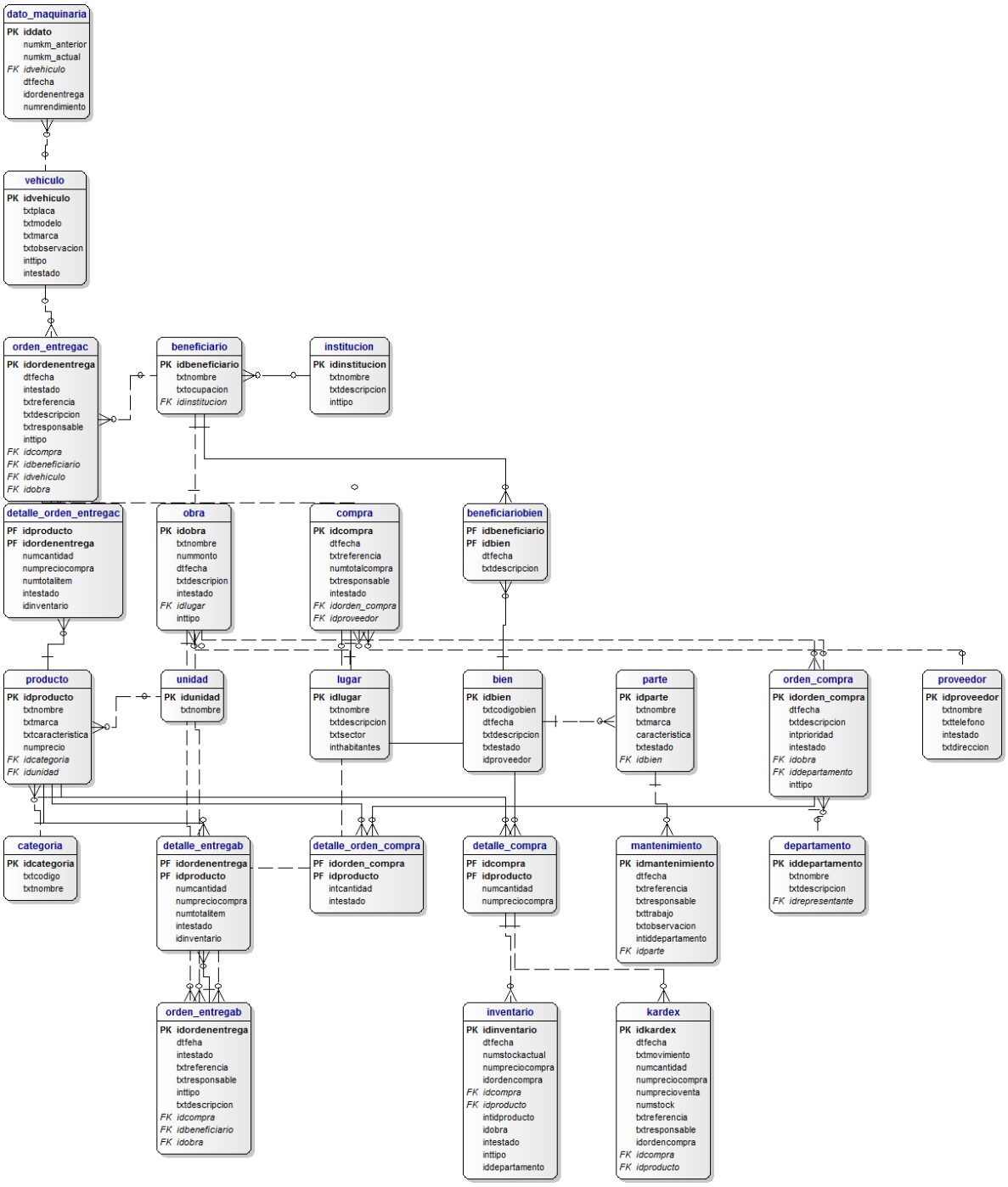


Figura V. 4. Diagrama físico de la base datos

Fuente: Autor

5.9 Fase de Desarrollo

Las versiones permiten a los clientes tener una visión clara de la aplicación desarrollada.

5.9.1Nomenclatura y Estándares

Se detallan a continuación.

Tabla V. 1. Nomenclatura y estándares

ARCHIVO	**EXTENSIÓN**
Programas en Java	fileName.jsp
Imágenes	Image.png
Clases	camelCase.java

Fuente: Autor

JAVA SERVER PAGES (JSP)

Esta tecnología es un desarrollo de la compañía Sun Microsystems. La Especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la Especificación JSP 2.1.

Las JSP's permiten la utilización de código Java mediante scripts. Además, es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Bibliotecas de Etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas.

5.9.2 Capa de Datos

5.9.2.1 Diccionario de Datos

Tabla V. 2. Diccionario de Datos

Nombre Campo	**tipo dato**	**Longitud**	**Clave Primaria**	**Calculado**
	TABLA: BENEFICIARIO			
idBeneficiario	serial		X	No

txtNombre	character	100		No
txtOcupacion	character	100		No
idInstitucion	integer			No
	TABLA: BENEFICIARIOBIEN			
idBeneficiario	integer		X	No
idParte	integer		X	No
dtFecha	date			No
txtDescripcion	character	200		No
	TABLA: BIEN			
idbien	serial		X	No
txtCodigoBien	character	40		No
dtFecha	date			No
txtDescripcion	character	200		No
txtEstado	character	40		No
idProveedor	integer			No
	TABLA: CATEGORÍA			
idcategoria	serial		X	No
txtCodigo	character	100		No
txtNombre	character	100		No
	TABLA: COMPRA			
idCompra	serial		X	No
dtfecha	date			No
txtReferencia	character	40		No
numTotalCompra	numeric	9,2		No
txtResposanble	character	100		No
intEstado	integer			No

idOrdenCompra	integer			No
Idproveedor	integer			No
	TABLA: DATOMAQUINARIA			
idDatoMaquinaria	serial		X	No
numkmAnterior	numeric	9,2		No
numkmActual	numeric	9,2		No
idVehiculo	integer			No
dtFecha	date			No
idOrdenEntrega	integer			No
numRendimiento	numeric	9,2		No
	TABLA: DEPARTAMENTO			
idDepartamento	serial		X	No
txtNombre	character	100		No
txtDescripcion	character	100		No
idRepresentante	integer			No
	TABLA: DETALLECOMPRA			
idCompra	integer		X	No
idProducto	integer		X	No
Numcantidad	numeric	9,2		No
numPrecioCompra	numeric	9,2		No
	TABLA: DETALLEENTREGAB			
idOrdenEntrega	integer		X	No
idProducto	integer		X	No
numCantidad	numeric	9,2		No
numPrecioCompra	numeric	9,2		Si
numTotalItem	numeric	9,2		Si

intEstado	integer			No
idInventario	integer			No
	TABLA:DETALLEORDENCOMPRA			
idOrdenCompra	integer		X	No
idProducto	integer		X	No
intCantidad	integer			No
intEstado	integer			No
	TABLA: DETALLEORDENENTREGAC			
idProducto	integer		X	No
idOrdenEntrega	integer		X	No
numCantidad	numeric	9,2		No
numPrecioCompra	numeric	9,2		Si
numTotalItem	numeric	9,2		Si
intestado	integer			No
idInventario	integer			No
	TABLA: INSTITUCION			
idInstitucion	serial		X	No
txtNombre	character	100		No
txtDescripcion	character	100		No
intTipo	integer			No
	TABLA: INVENTARIO			
idInventario	serial		X	No
dtFecha	date			No
numStockActual	numeric	9,2		No
numprecioCompra	numeric	9,2		No
idOrdenCompra	integer			No

idCompra	integer			No
idProducto	integer			No
idObra	integer			No
intestado	integer			No
intTipo	integer			No
idDepartamento	integer			No
	TABLA: LUGAR			
idLugar	serial		X	No
txtNombre	character	100		No
txtDescripcion	character	100		No
txtSector	character	100		No
intHabitantes	integer			No
	TABLA: MANTENIMIENTO			
idMantenimiento	serial		X	No
dtFecha	date			No
txtReferencia	character	40		No
txtResponsable	character	40		No
txtTrabajo	character	200		No
txtObservacion	character	200		No
idDepartamento	integer			No
idParte	integer			No
	TABLA: OBRA			
idObra	serial		X	No
txtNombre	character	100		No
numMonto	numeric	9,2		No
dtFecha	date			No

txtDescripcion	character	100		No
intestado	integer			No
idLugar	integer			No
	TABLA: ORDENCOMPRA			
idOrdenCompra	serial		X	No
dtFecha	date			No
txtDescripcion	character	150		No
intPrioridad	integer			No
intestado	integer			No
idObra	integer			No
idDepartamento	integer			No
intTipo	integer			No
	TABLA: PARTE			
idParte	serial		X	No
txtNombre	character	100		
txtMarca	character	100		No
txtCaracteristica	character	100		No
txtEstado	character	40		No
idBien	integer			No
	TABLA: PRODUCTO			
idProducto	serial		X	No
txtNombre	character	100		No
txtMarca	character	100		No
txtCaracteristica	character	100		No
numPrecio	numeric	9,2		No
idCategoria	integer			No

idUnidad	integer			No
	TABLA: PROVEEDOR			
idProveedor	serial		X	No
txtNombre	character	100		No
txtTelefono	character	10		No
intestado	integer			No
txtDirecccion	character	100		No
	TABLA: REPRESENTANTE			
idRepresentante	serial		X	No
txtNombre	character	100		No
dtFecha	date			No
intestado	integer			No
	TABLA: UNIDAD			
idUnidad	serial		X	No
txtNombre	character	40		No
	tabla: usuario			
idUsuario	serial		X	No
txtNombre	character	100		No
txtCargo	character	40		No
txtUsuario	character	40		No
txtClave	character	40		No
intTipo	integer			No
idDepartamento	integer			No
	TABLA: VEHICULO			
idVehiculo	serial		X	No
txtPlaca	character	10		No

txtModelo	character	200		No
txtMarca	character	200		No
txtObservacion	character	100		No
intTipo	integer			No
intestado	integer			no

Fuente: Autor

5.9.2.2 Script de la Base de Datos

```
-- Database: "sgb_Municipio"

-- DROP DATABASE "sgb_Municipio";

CREATE DATABASE "sgb_Municipio"

  WITH OWNER = usermunicipio

       ENCODING = 'UTF8'

       TABLESPACE = pg_default

       LC_COLLATE = 'Spanish_Ecuador.1252'

       LC_CTYPE = 'Spanish_Ecuador.1252'

       CONNECTION LIMIT = -1;


/* ---------------------------------------------------------------------- */

/* Script generated with: DeZign for Databases V7.1.2                     */

/* Target DBMS:           PostgreSQL 9                                    */

/* Project file:          sgb.dez                                         */

/* Project name:                                                          */

/* Author:                                                                */

/* Script type:           Database creation script                        */

/* Created on:            2012-05-24 13:37                                */

/* ---------------------------------------------------------------------- */
```

```
/* ---------------------------------------------------------------------- */
/* Tables                                                                 */
/* ---------------------------------------------------------------------- */

/* ---------------------------------------------------------------------- */
/* Add table "categoria"                                                  */
/* ---------------------------------------------------------------------- */

CREATE TABLE categoria (
    idCategoria SERIAL  NOT NULL,
    txtCodigo CHARACTER(100),
    txtNombre CHARACTER(100),
    CONSTRAINT PK_categoria PRIMARY KEY (idCategoria)
);

/* ---------------------------------------------------------------------- */
/* Add table "Unidad"                                                     */
/* ---------------------------------------------------------------------- */

CREATE TABLE Unidad (
    idUnidad SERIAL  NOT NULL,
    txtNombre CHARACTER(100),
    CONSTRAINT PK_Unidad PRIMARY KEY (idUnidad)
);

/* ---------------------------------------------------------------------- */
```

```
/* Add table "producto"                                          */
/* ---------------------------------------------------------------- */

CREATE TABLE producto (
    idProducto SERIAL  NOT NULL,
    idCategoria INTEGER  NOT NULL,
txtNombre CHARACTER(100),
    txtMarca CHARACTER(100),
    txtCaracteristica CHARACTER(100),
numPrecio NUMERIC(9,2),
    idUnidad INTEGER  NOT NULL,
    CONSTRAINT PK_producto PRIMARY KEY (idProducto)
);

/* ---------------------------------------------------------------- */
/* Add table "lugar"                                             */
/* ---------------------------------------------------------------- */

CREATE TABLE lugar (
    idLugar SERIAL  NOT NULL,
    txtNombre CHARACTER(100),
    txtDescripcion CHARACTER(100),
    txtSector CHARACTER(100),
    intHabitantes INTEGER,
    CONSTRAINT PK_lugar PRIMARY KEY (idLugar)
);
```

```
/* ---------------------------------------------------------------------- */
/* Add table "obra"                                                       */
/* ---------------------------------------------------------------------- */

CREATE TABLE obra (
    idObra SERIAL  NOT NULL,
    txtNombre CHARACTER VARYING(100) DEFAULT 'null'  NOT NULL,
    numMonto NUMERIC(9,2) DEFAULT 0,
    dtFecha DATE,
    txtDescripion CHARACTER VARYING(100),
    intEstado INTEGER DEFAULT 1,
    idLugar INTEGER  NOT NULL,
    CONSTRAINT PK_obra PRIMARY KEY (idObra)
);

COMMENT ON COLUMN obra.intEstado IS '1 = activo 0 = inactivo';

/* ---------------------------------------------------------------------- */
/* Add table "inventario"                                                 */
/* ---------------------------------------------------------------------- */

CREATE TABLE inventario (
    idInventario SERIAL  NOT NULL,
    idProducto INTEGER  NOT NULL,
    dtFecha DATE,
    numStockactual NUMERIC(9,2),
    numPreciocompra MONEY,
```

```
    CONSTRAINT PK_inventario PRIMARY KEY (idInventario)
);

/* ---------------------------------------------------------------------- */
/* Add table "kardex"                                                     */
/* ---------------------------------------------------------------------- */

CREATE TABLE kardex (
    idKardex SERIAL  NOT NULL,
    idProducto INTEGER  NOT NULL,
dtfecha DATE,
txtMovimiento CHARACTER VARYING(40),
    numCantidad INTEGER,
    numPreciocompra NUMERIC(9,2),
numPrecioventa NUMERIC(9,2),
    numStock NUMERIC(9,2),
    txtReferencia CHARACTER VARYING(40),
    txtResponsable CHARACTER VARYING(100),
    CONSTRAINT PK_kardex PRIMARY KEY (idKardex)
);

/* ---------------------------------------------------------------------- */
/* Add table "representante"                                              */
/* ---------------------------------------------------------------------- */

CREATE TABLE representante (
    idRepresentante SERIAL  NOT NULL,
```

```
    txtNombre CHARACTER VARYING(100)  NOT NULL,
    dtFecha DATE,
    CONSTRAINT PK_representante PRIMARY KEY (idRepresentante)
);

/* ---------------------------------------------------------------------- */
/* Add table "proveedor"                                                  */
/* ---------------------------------------------------------------------- */

CREATE TABLE proveedor (
    idProveedor SERIAL  NOT NULL,
    txtNombre CHARACTER VARYING(100),
txtTelefono CHARACTER(9),
    intEstado INTEGER DEFAULT 1,
txtDireccion CHARACTER VARYING(100),
    CONSTRAINT PK_proveedor PRIMARY KEY (idProveedor)
);

COMMENT ON COLUMN proveedor.intEstado IS '1=activo 2=no activo';

/* ---------------------------------------------------------------------- */
/* Add table "vehiculo"                                                   */
/* ---------------------------------------------------------------------- */

CREATE TABLE vehiculo (
    idVehiculo SERIAL  NOT NULL,
    txtPlaca CHARACTER(10),
```

```
    txtModelo CHARACTER(20),
    txtMarca CHARACTER(20),
    txtObservacion CHARACTER(100),
    CONSTRAINT PK_vehiculo PRIMARY KEY (idVehiculo)
);

/* ---------------------------------------------------------------------- */
/* Add table "dato_maquinaria"                                            */
/* ---------------------------------------------------------------------- */

CREATE TABLE dato_maquinaria (
    idDato SERIAL  NOT NULL,
    numKm_anterior NUMERIC(9,2),
    numKm_actual NUMERIC(9,2),
    numH_actual NUMERIC(9,2),
    numH_anterior NUMERIC(9,2),
    CONSTRAINT PK_dato_maquinaria PRIMARY KEY (idDato)
);

/* ---------------------------------------------------------------------- */
/* Add table "institucion"                                                */
/* ---------------------------------------------------------------------- */

CREATE TABLE institucion (
    idInstitucion SERIAL  NOT NULL,
    txtNombre CHARACTER VARYING(100),
    txtDescripcion CHARACTER VARYING(100),
```

```
    intTipo INTEGER,
    CONSTRAINT PK_institucion PRIMARY KEY (idInstitucion)
);

COMMENT ON COLUMN institucion.intTipo IS '0=publico 1=privado';

/* ---------------------------------------------------------------------- */
/* Add table "usuario"                                                    */
/* ---------------------------------------------------------------------- */

CREATE TABLE usuario (
    idusuario SERIAL  NOT NULL,
    txtNombre CHARACTER VARYING(100)  NOT NULL,
    txtCargo CHARACTER VARYING(40)  NOT NULL,
    txtUsuario CHARACTER VARYING(40)  NOT NULL,
    txtClave CHARACTER VARYING(40)  NOT NULL,
    intTipo INTEGER  NOT NULL,
    CONSTRAINT PK_usuario PRIMARY KEY (idusuario)
);

COMMENT ON COLUMN usuario.intTipo IS '1=Administrador';

/* ---------------------------------------------------------------------- */
/* Add table "departamento"                                               */
/* ---------------------------------------------------------------------- */

CREATE TABLE departamento (
```

```
    idDepartamento SERIAL  NOT NULL,
    txtNombre CHARACTER VARYING(100)  NOT NULL,
    txtDescripcion CHARACTER VARYING(100),
    idRepresentante INTEGER  NOT NULL,
    CONSTRAINT PK_departamento PRIMARY KEY (idDepartamento)
);

/* ---------------------------------------------------------------------- */
/* Add table "beneficiario"                                               */
/* ---------------------------------------------------------------------- */

CREATE TABLE beneficiario (
    idBeneficiario SERIAL  NOT NULL,
    txtNombre CHARACTER VARYING(100)  NOT NULL,
    txtOcupacion CHARACTER VARYING(100),
    idInstitucion INTEGER  NOT NULL,
    CONSTRAINT PK_beneficiario PRIMARY KEY (idBeneficiario)
);

/* ---------------------------------------------------------------------- */
/* Add table "orden_compra"                                               */
/* ---------------------------------------------------------------------- */

CREATE TABLE orden_compra (
    idOrden_compra SERIAL  NOT NULL,
    dtFecha DATE,
    txtDescripcion CHARACTER VARYING(40),
```

```
    intPrioridad INTEGER DEFAULT 0,
    intEstado INTEGER DEFAULT 1,
    idObra INTEGER  NOT NULL,
    idDepartamento INTEGER  NOT NULL,
    CONSTRAINT PK_orden_compra PRIMARY KEY (idOrden_compra)
);
COMMENT ON COLUMN orden_compra.intPrioridad IS '1=baja 2=media 3=alta';

COMMENT ON COLUMN orden_compra.intEstado IS '1=activo 0=inactivo
2=pendiente';

/* ---------------------------------------------------------------------- */
/* Add table "detalle_orden_compra"                                       */
/* ---------------------------------------------------------------------- */

CREATE TABLE detalle_orden_compra (
    idOrden_compra INTEGER  NOT NULL,
idProducto INTEGER  NOT NULL,
    intCantidad INTEGER DEFAULT 0  NOT NULL,
intEstado INTEGER DEFAULT 0,
    PRIMARY KEY (idOrden_compra, idProducto)
);

COMMENT ON COLUMN detalle_orden_compra.intEstado IS '0=adquirido
1=pendiente';

/* ---------------------------------------------------------------------- */
```

```
/* Add table "compra"                                                    */
/* ---------------------------------------------------------------------- */



CREATE TABLE compra (
idCompra SERIAL  NOT NULL,
    dtFecha DATE,
    txtReferencia CHARACTER VARYING(40),
    numTotalcompra NUMERIC(9,2),
    txtResponsable CHARACTER VARYING(100),
    intEstado INTEGER,
    idOrden_compra INTEGER  NOT NULL,
idProveedor INTEGER  NOT NULL,
    CONSTRAINT PK_compra PRIMARY KEY (idCompra)
);
COMMENT ON COLUMN compra.intEstado IS '2=en proceso 1=activo 0=pendiente';
/* ---------------------------------------------------------------------- */
/* Add table "detalle_compra"                                            */
/* ---------------------------------------------------------------------- */
CREATE TABLE detalle_compra (
idCompra INTEGER  NOT NULL,
    idProducto INTEGER  NOT NULL,
    numCantidad NUMERIC(9,2),
    numPreciocompra NUMERIC(9,2),
    CONSTRAINT PK_detalle_compra PRIMARY KEY (idCompra, idProducto)
);
```

```
/* ---------------------------------------------------------------------- */
/* Add table "orden_entregab"                                             */
/* ---------------------------------------------------------------------- */

CREATE TABLE orden_entregab (
idOrdenentrega SERIAL  NOT NULL,
    dtFeha DATE,
    intEstado INTEGER,
    txtReferencia CHARACTER VARYING(40),
    txtResponsable CHARACTER VARYING(40),
    intTipo INTEGER,
    txtDescripcion CHARACTER VARYING(100),
    idCompra INTEGER,
    idBeneficiario INTEGER  NOT NULL,
    idObra INTEGER  NOT NULL,
    CONSTRAINT PK_orden_entregab PRIMARY KEY (idOrdenentrega)
);

COMMENT ON COLUMN orden_entregab.idOrdenentrega IS '1=activo 0=inactivo';

COMMENT ON COLUMN orden_entregab.intEstado IS '1=pendiente 2=activo
3=inactivo';

COMMENT ON COLUMN orden_entregab.intTipo IS '2=bodega';

/* ---------------------------------------------------------------------- */
/* Add table "detalle_entregab"                                           */
```

```
/* ---------------------------------------------------------------------- */

CREATE TABLE detalle_entregab (
    idOrdenentrega SERIAL  NOT NULL,
    idProducto INTEGER  NOT NULL,
    numCantidad NUMERIC(9,2) DEFAULT 0,
    numPreciocompra NUMERIC(9,2),
    numTotalitem NUMERIC(9,2),
    intEstado INTEGER,
    CONSTRAINT PK_detalle_entregab PRIMARY KEY (idOrdenentrega, idProducto)
);

COMMENT ON COLUMN detalle_entregab.intEstado IS '0=pendiente 1=entregado';

/* ---------------------------------------------------------------------- */
/* Add table "orden_entregac"                                             */
/* ---------------------------------------------------------------------- */


CREATE TABLE orden_entregac (
    idOrdenentrega SERIAL  NOT NULL,
    dtFecha DATE,
    intEstado INTEGER DEFAULT 0,
    txtReferencia CHARACTER VARYING(40),
    txtDescripcion CHARACTER VARYING(100),
    txtResponsable CHARACTER VARYING(100),
    inttipo INTEGER,
```

```
    idCompra INTEGER,
    idBeneficiario INTEGER,
    idDato INTEGER,
    idVehiculo INTEGER,
    idObra INTEGER  NOT NULL,
    CONSTRAINT PK_orden_entregac PRIMARY KEY (idOrdenentrega)
);

COMMENT ON COLUMN orden_entregac.intEstado IS '1=combustible';

COMMENT ON COLUMN orden_entregac.inttipo IS '0=inactivo 1=activo';

/* ---------------------------------------------------------------------- */
/* Add table "detalle_orden_entregac"                                     */
/* ---------------------------------------------------------------------- */

CREATE TABLE detalle_orden_entregac (
    idProducto INTEGER  NOT NULL,
    idOrdenentrega INTEGER  NOT NULL,
    numCantidad NUMERIC(9,2) DEFAULT 0,
    numPreciocompra NUMERIC(9,2),
    numTotalitem NUMERIC(9,2),
    intEstado INTEGER DEFAULT 0,
    CONSTRAINT PK_detalle_orden_entregac PRIMARY KEY (idProducto,
idOrdenentrega)
);
```

```
COMMENT ON COLUMN detalle_orden_entregac.intEstado IS '0=pendiente
1=entegado';

/* ---------------------------------------------------------------------- */
/* Foreign key constraints                                                */
/* ---------------------------------------------------------------------- */

ALTER TABLE producto ADD CONSTRAINT categoria_producto
   FOREIGN KEY (idCategoria) REFERENCES categoria (idCategoria);

ALTER TABLE producto ADD CONSTRAINT Unidad_producto
   FOREIGN KEY (idUnidad) REFERENCES Unidad (idUnidad);

ALTER TABLE obra ADD CONSTRAINT lugar_obra
   FOREIGN KEY (idLugar) REFERENCES lugar (idLugar);

ALTER TABLE orden_compra ADD CONSTRAINT obra_orden_compra
   FOREIGN KEY (idObra) REFERENCES obra (idObra);

ALTER TABLE orden_compra ADD CONSTRAINT departamento_orden_compra
   FOREIGN KEY (idDepartamento) REFERENCES departamento (idDepartamento);

ALTER TABLE detalle_orden_compra ADD CONSTRAINT
orden_compra_detalle_orden_compra
   FOREIGN KEY (idOrden_compra) REFERENCES orden_compra
(idOrden_compra);
```

```
ALTER TABLE detalle_orden_compra ADD CONSTRAINT
producto_detalle_orden_compra
FOREIGN KEY (idProducto) REFERENCES producto (idProducto);

ALTER TABLE inventario ADD CONSTRAINT producto_inventario
   FOREIGN KEY (idProducto) REFERENCES producto (idProducto);

ALTER TABLE kardex ADD CONSTRAINT producto_kardex
   FOREIGN KEY (idProducto) REFERENCES producto (idProducto);

ALTER TABLE departamento ADD CONSTRAINT representante_departamento
   FOREIGN KEY (idRepresentante) REFERENCES representante (idRepresentante);

ALTER TABLE compra ADD CONSTRAINT orden_compra_compra
FOREIGN KEY (idOrden_compra) REFERENCES orden_compra (idOrden_compra);

ALTER TABLE compra ADD CONSTRAINT proveedor_compra
   FOREIGN KEY (idProveedor) REFERENCES proveedor (idProveedor);

ALTER TABLE detalle_compra ADD CONSTRAINT compra_detalle_compra
   FOREIGN KEY (idCompra) REFERENCES compra (idCompra);

ALTER TABLE detalle_compra ADD CONSTRAINT producto_detalle_compra
   FOREIGN KEY (idProducto) REFERENCES producto (idProducto);

ALTER TABLE orden_entregab ADD CONSTRAINT compra_orden_entregab
   FOREIGN KEY (idCompra) REFERENCES compra (idCompra);
```

```
ALTER TABLE orden_entregab ADD CONSTRAINT beneficiario_orden_entregab
   FOREIGN KEY (idBeneficiario) REFERENCES beneficiario (idBeneficiario);

ALTER TABLE orden_entregab ADD CONSTRAINT obra_orden_entregab
   FOREIGN KEY (idObra) REFERENCES obra (idObra);

ALTER TABLE beneficiario ADD CONSTRAINT institucion_beneficiario
   FOREIGN KEY (idInstitucion) REFERENCES institucion (idInstitucion);

ALTER TABLE detalle_entregab ADD CONSTRAINT
orden_entregab_detalle_entregab
   FOREIGN KEY (idOrdenentrega) REFERENCES orden_entregab (idOrdenentrega);

ALTER TABLE detalle_entregab ADD CONSTRAINT producto_detalle_entregab
   FOREIGN KEY (idProducto) REFERENCES producto (idProducto);

ALTER TABLE orden_entregac ADD CONSTRAINT compra_orden_entregac
   FOREIGN KEY (idCompra) REFERENCES compra (idCompra);

ALTER TABLE orden_entregac ADD CONSTRAINT beneficiario_orden_entregac
   FOREIGN KEY (idBeneficiario) REFERENCES beneficiario (idBeneficiario);

ALTER TABLE orden_entregac ADD CONSTRAINT dato_maquinaria_orden_entregac
   FOREIGN KEY (idDato) REFERENCES dato_maquinaria (idDato);

ALTER TABLE orden_entregac ADD CONSTRAINT vehiculo_orden_entregac
```

FOREIGN KEY (idVehiculo) REFERENCES vehiculo (idVehiculo);

ALTER TABLE orden_entregac ADD CONSTRAINT obra_orden_entregac

FOREIGN KEY (idObra) REFERENCES obra (idObra);

ALTER TABLE detalle_orden_entregac ADD CONSTRAINT orden_entregac_detalle_orden_entregac

FOREIGN KEY (idOrdenentrega) REFERENCES orden_entregac (idOrdenentrega);

ALTER TABLE detalle_orden_entregac ADD CONSTRAINT producto_detalle_orden_entregac

FOREIGN KEY (idProducto) REFERENCES producto (idProducto);

5.9.2.3 Implementación de la base de Datos

Examinar Objetos de la Base de Datos

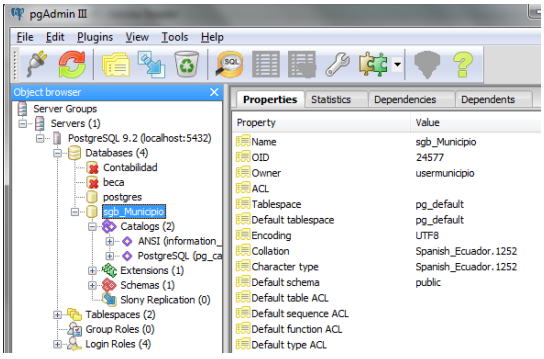


Figura.V.15. Objetos de la base de datos

Fuente: Autor

Examinar todas las tablas de la base de datos

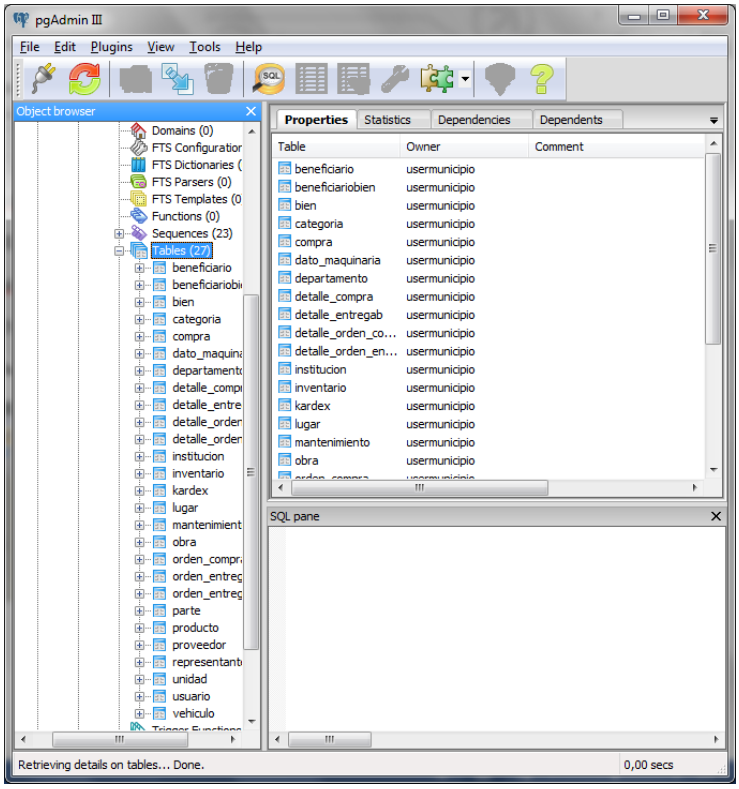


Figura.V.16. Tablas de la base de datos

Fuente: Autor

Estructura de una Tabla

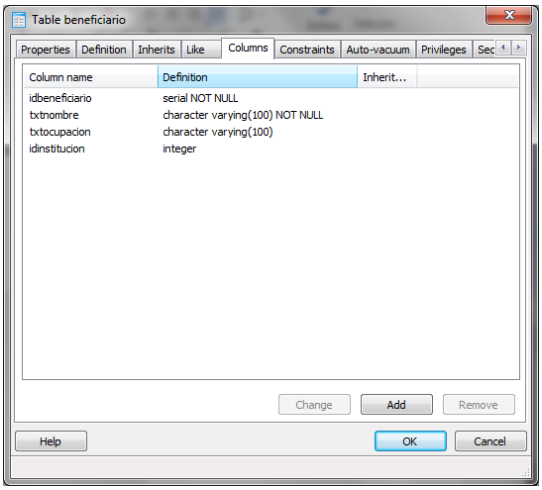


Figura.V.17. Estructura de una Tabla

Fuente: Autor

5.10. Fase de Estabilización

Esta fase permite validar los requerimientos de la solución implementada. El desarrollo de la solución del proyecto ha sido completado y realizamos las pruebas correspondientes.

5.10.1 Revisión general del Sistema

5.10.1.1 Código Fuente

El código fuente se presenta en el anexo

Documentación de la instalación

Para ayudar al uso y manipulación del mismo se presenta los siguientes manuales:

- Manual Técnico
- Manual Usuario

5.11Fase de Instalación

5.11.1 Tareas a realizar

La aplicación ha sido finalizada y se procede a la entrega al Responsable de la unidad de Sistema del GAD Municipal Arosemena Tola.

CONCLUSIONES

1. La forma más común para hacer uso de un ORM es usar una base de datos relacional.
2. En la actualidad existen ORMs que con el paso del tiempo han madurado y solucionan de una manera transparente, eficiente el problema de la persistencia de datos y modelo de dominios completamente orientada a objetos. Con esta investigación se demostró que Hibernate alcanzó un 98% de eficiencia por lo que se determina que es el framework más adecuado para desarrollar una aplicación web empresarial en ambiente J2EE; por otro lado Eclipselink obtuvo una calificación de 78,43%.
3. Es necesario conocer bien el funcionamiento de las tecnologías que se utilizaran en el desarrollo de una aplicación. En el caso del Framework de Hibernate he visto que el rendimiento está directamente relacionada con la configuración y el diseño adecuado de la base de datos. Respecto al manejo de consultas Hibernate proporciona del lenguaje “HQL” que lo hace multi-motor de base de datos lo cual lo hace apetecible frente a los demás ORMs.
4. El servicio implementado en el GAD Municipal de Arosemena Tola para la gestión y control de bienes garantiza el correcto tratamiento de la información, haciendo uso de los recursos existentes.

5. Durante el curso de esta investigación, Hibernate no solo demostró ser una tecnología más eficiente, sino que también más fácil de usar en comparación a EclipseLink, con la suficiente información por parte de sus desarrolladores como JBoss y se puede acceder a ella de manera gratuita.

RECOMENDACIONES

1. Es de vital importancia estudiar un ORM antes de hacer uso de la misma porque el rendimiento depende directamente de la elección del mismo.
2. Los sistemas debemos construirlos en función de patrones, estándares y reutilizando al máximo el código, ya que nos permitirá centrarnos en la lógica de negocio, reducir los tiempos de desarrollo y aumentar la calidad del mismo.
3. Es necesario que el GAD municipal defina políticas que regulen el desarrollo de sistemas de manera que aproveche al máximo la tecnología e infraestructura adquirida.

RESUMEN

Se analizó la tecnología Hibernate como Framework de persistencia en el desarrollo de aplicaciones web. Caso Práctico: Control y gestión de bienes del Gobierno Autónomo Descentralizo Municipal del cantón Arosemena Tola, Provincia Napo.

Para el desarrollo de esta investigación se utilizó el método de investigación Científico con el fin de levantar, recopilar información, analizar e interpretar los resultados que permitieron la comprobación de la hipótesis para ello se utilizaron recursos hardware, software para la implementación del sistema y ofimático como son las encuestas para su posterior tabulación y aplicación del método estadístico T-Student; además se hizo uso del método descriptivo para describir las características más sobresalientes de la tecnología Hibernate.

Como resultados del análisis y comparación de las tecnologías de Mapeo Objeto Relacional (ORM) Hibernate y EclipseLink, se mostro que Hibernate alcanzo el 98% de eficiencia frente a 78,43% alcanzado por EclipseLink.

Como se demostró en el estudio que la tecnología Hibernate es de gran ayuda para los desarrolladores de aplicaciones basados en la tecnología J2EE. Se concluye que reduce significativamente el tiempo de desarrollo de la aplicación, consume menos recursos, permite adaptar código de acuerdo a la necesidad del usuario, puede

integrarse con una gran variedad de Frameworks, posee una excelente documentación y se acopla muy bien a la arquitectura Modelo Vista Controlador.

Se recomienda el uso de este Framework para el desarrollo de aplicaciones web empresariales puesto que ofrece grandes ventajas al reducir tiempo de desarrollo, eliminando muchos problemas con el manejo de base de datos relacionales.

ABSTRACT

The Hibernate Technology wiht persistence framework was analyzed in web application develoment. Case Study: Control and management of the Autonomous Descentralized Municipal Government assets, Arosemena Tola Conton, Napo Province.

For this research development the scientific method was developed in order to gather and collect information, analyze and interpret results which allowed the hypothesis testing, hardware resources, software resources for the system and office implementation such as surveys for subsequent tabulation and statistic T-student method application were used; the descriptive method was also used to describe the most relevant features of the Hibernate technology.

As result of the analysis and comparison of the Object Relational Mapping (ORM), Hibernate and EclipseLink technologies it was demonstrated that the Hibernate reached 98% efficiency compared to 78,43% reached by EclipseLink.

As it is demonstrated in the study, the Hibernate technology is helpful for applications based on J2EE technology developers. It is concluded that it reduces significantly the applicacionts development time, it consumers fewer resources, it allows adapting codes according to the user needs, it can be integrated with a variety of frameworks, it

has an excellent documentation and it fits well to the View Controller Model architecture.

This framework use is recommended for enterprise web applications development because offers great advantages reducing development time, eliminating many problems with the related database management.

GLOSARIO

Anotación Java: Forma de añadir metadatos al código fuente Java que están disponibles para la aplicación en tiempo de ejecución.

API: Conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Representa una interfaz de comunicación entre componentes software.

Aplicación web: Aplicación informática que los usuarios utilizan accediendo a un servidor web a través de Internet o de una intranet.

Arquitectura: Representación abstracta de los componentes de un sistema y su comportamiento

Byte: Unidad de información formada por ocho bits.

Clase: Definición de un objeto.

DAO: componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

EclipseLink: Implementación de JPA.

EntityManager: Interfaz que define los métodos que son usados para interactuar con el contexto de persistencia.

Facade: Patrón de Diseño, sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas.

Framework de Persistencia: Componente de software encargado de traducir entre objetos y registros (de la base de datos relacional). Es decir es el encargado de que el programa y la base de datos se “entiendan”.

Hibernate: Herramienta de Mapeo objeto-relacional para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

HQL (Lenguaje de Consulta Hibernate): Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Al mismo tiempo que una API para construir las consultas programáticamente

Java: Lenguaje de Programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90.

JCP (Java Community Process): Organismo que crea y mantiene las especificaciones para Java.

JDBC: Java DataBase Connectivity es el API de Java que define como una aplicación cliente accederá a una base de datos, independientemente del motor de base de datos al que accedamos.

JPA. Indica que el proveedor de persistencia se hará cargo de la gestión de transacciones.

JPA: Framework de Persistencia.

JSR (Java Specification Request): Son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java.

Mapping: Proceso de conectar objetos/atributos a tablas/columnas.

Metadatos: Descripciones estructuradas y opcionales que están disponibles de forma pública para ayudar a localizar objetos.

MVC: Patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Objeto persistente: Objeto que sobrevive a la ejecución del proceso que lo creó.

Objeto transiente: Objeto que deja de existir cuando el proceso que lo creó termina su ejecución.

Objeto: Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos). Se corresponde con los objetos reales del mundo que nos rodea, o a **objetos internos del sistema (del programa).** Es una instancia a una clase.

ORM (Object/Relational Mapping): Técnica que realiza la transición de una representación de los datos de un modelo relacional a un modelo orientado a objetos y viceversa.

Patrón de diseño: Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

Persistencia: Capacidad de almacenar y recuperar el estado de los objetos, de forma que sobrevivan a los procesos que los manipulan.

POJO (Plain Old Java Object): Simple clase Java que tiene métodos get y set para cada uno de los atributos.

RESOURCE_LOCAL: Tipo de transacción para aplicación con soporte

Serialización: Secuencia de bytes escrita en un fichero en disco.

SQL: Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

BIBLIOGRAFÍA

1. **BauerC**; **GavinK**. Hibernate in Action Practical Object/Relational Mapping. Greenwich - Estados Unidos, Manning, © 2004. pp.29-50.
2. **BauerC**; **GavinK**. Java Persistence with Hibernate. s.l., Manning, © 2007. pp.51-190.
3. **James E; Tim O** y otros. Harnessing Hibernate. California-Estados Unidos, O'Reilly Media, ©2008. pp.52-90; pp.137-139.
4. **Mike K; Merrick S.** Pro EJB 3: Java Persistence API. 2.ed. New York-Estados Unidos, Apress, © 2010. pp.51-160.
5. **Richard S**. Java Persistence for Relational Databases. New York-Estados Unidos, Apress, © 2003. pp.30-33.
6. **Scott W**. Mapping objects to relational databases: O/R mapping in detail. Londres-Inglaterra, Ambler, © 2006. pp.30-31

BIBLIOGRAFÍA DE INTERNET

7. **ADOPTING A JAVA PERSISTENCE FRAMEWORK: WHICH, WHEN, AND WHAT?**

   http://today.java.net/pub/a/today/2007/12/18/adopting-java- persistence-Framework.html

   2012/10/20

8. **APLICACIÓN WEB CON HIBERNATE**

   http://www.ooscarr.com/nerd/elblog/2009/12/ejemplo-de-aplicacion-web-con-Hibernate.php

   2012/09/28

9. **ARQUITECTURA DE SOFTWARE INTRODUCCIÓN A LA JAVA PERSISTENCE API**

   http://sophia.javeriana.edu.co/~javila/pregrado/arquitectura/JPA.pdf.

   2012/08/28

10. **BASES DE OBJETOS**

    http://www.db4o.com/espanol/db4o%20Whitepaper%20%20Bases%20de%20Objetos.pdf

    2012/06/24

11. **ECLIPSELINK**

    http://www.eclipse.org/eclipselink/

    2012/10/25

12. **EJEMPLO DE HIBERNATE**

    http://chuwiki.chuidiang.org/index.php?title=Ejemplo_sencillo_con_Hibernate

    2012/08/28

13. EJEMPLO DE HIBERNATE 3 Y ANOTACIONES JPA BÁSICO

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=AnotacionesEJB3

2012/09/13

14. JAVA PERSISTENCE API (JPA)

http://www.coplec.org/?q=book/export/html/240

2012/09/30

15. MAPEO OBJETO RELACIONAL

http://k7k0.apihuella.com/tesis/mapeo_objeto_relacional.htm

2012/06/24

http://pizarropablo.googlepages.com/ORM-ObjectRelationalMapping-PizarroP.pdf

2012/08/28

16. MODELO VISTA CONTROLADOR

http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

2012/10/20

17. ORM

http://www.iit.upcomillas.es/pfc/resumenes/450955e7368ca.pdf

2012/06/24

18. PERSISTENCIA

http://java.sun.com/developer/technicalArticles/J2SE/Desktop/persistenceapi/?feed=JSC

2012/09/26

19. PERSISTENCIA BÁSICA EN JAVA

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=PersistenciaJava

2012/03/01

20. PERSISTENCIA DE DATOS EN JAVA

http://www.slideshare.net/ikercanarias/persistencia-de-datos-en-java
2012/10/15

21. PERSISTENCIA DE OBJETOS

http://es.wikipedia.org/wiki/Persistencia_de_objetos
2012/04/25

22. PERSISTENCIA DE OBJETOS JAVA: EL CAMINO HACIA HIBERNATE

http://www.programacion.com/articulo/persistencia_de_objetos_java:_el_camino_hacia_Hibernate_251
2012/09/30

ANEXOS

ANEXO 1. ENCUESTAS

ENCUESTA

Objetivo: *La presente encuesta tiene por objetivo identificar el mecanismo de trabajo en el GAD Municipal respecto a la solicitud de materiales de bodega y su utilización, para tener claridad de su situación actual.*

Instructivo: *Marque con una X el casillero de su elección. La encuesta es anónima no requiere su identificación.*

Pregunta 1: ¿La institución cuenta con un Sistema de Solicitud y Control de Bienes que interactúe con todos los departamentos?

_____Si
_____No

Pregunta 2: ¿Cuál es el grado de satisfacción con la forma manual con la que se viene

llevando el control y uso de bienes?

_____Muy Satisfactorio
_____Satisfactorio
_____Bueno
_____Malo
_____Pésimo

Pregunta 3: ¿Cómo calificaría el proceso de solicitud de materiales a bodega en la

actualidad?

_____Muy Satisfactorio
_____Satisfactorio
_____Bueno
_____Malo
_____Pésimo

Pregunta 4: ¿Cómo calificaría usted la calidad de información sobre la disponibilidad

de materiales dada por los responsables de bodega?

_____Muy Satisfactorio

_____Satisfactorio

_____Bueno

_____Malo

_____Pésimo

Pregunta 5: ¿Usted conoce la disponibilidad de los materiales de una obra antes de generar una orden de entrega?

_____Siempre

_____Muy a menudo

_____A menudo

_____Casi nunca

_____Nunca

Pregunta 6: ¿El Departamento tiene conocimiento de las obras que se desarrollan en el cantón?

_____Si
_____No

Pregunta 7: ¿Cómo califica el proceso de entrega y control de combustibles en la institución?

_____Muy Satisfactorio

_____Satisfactorio

_____Bueno

_____Malo

_____Pésimo

Pregunta 8: ¿Existe información disponible sobre los equipos de cómputo para fines de administración y toma de decisiones?

_____Si
_____No

ENCUESTA

Objetivo: *La presente encuesta tiene por objetivo identificar el mecanismo de trabajo en el GAD Municipal respecto a la solicitud de materiales de bodega y su utilización, luego de la implementación del Sistema de Gestión de Bienes.*

Instructivo: *Marque con una X el casillero de su elección. La encuesta es anónima no requiere su identificación.*

Pregunta 1: ¿La institución cuenta con un Sistema de Solicitud y Control de Bienes

que interactúe con todos los departamentos?

_____Si
_____No

Pregunta 2: ¿Cuál es el grado de satisfacción con el funcionamiento del sistema SGB

desarrollado?

_____Muy Satisfactorio
_____Satisfactorio
_____Bueno
_____Malo
_____Pésimo

Pregunta 3: ¿Cómo calificaría el proceso de solicitud de materiales a bodega en la

actualidad?

_____Muy Satisfactorio
_____Satisfactorio
_____Bueno
_____Malo
_____Pésimo

Pregunta 4: ¿Cómo calificaría usted la calidad de información sobre la disponibilidad

de materiales dada por los responsables de bodega?

_____Muy Satisfactorio
_____Satisfactorio
_____Bueno
_____Malo
_____Pésimo

Pregunta 5: ¿Usted conoce la disponibilidad de los materiales de una obra antes de generar una orden de entrega?

_____Siempre

_____Muy a menudo

_____A menudo

_____Casi nunca

_____Nunca

Pregunta 6: ¿El Departamento tiene conocimiento de las obras que se desarrollan en el cantón?

_____Si
_____No

Pregunta 7: ¿Cómo califica el proceso de entrega y control de combustibles en la institución?

_____Muy Satisfactorio
_____Satisfactorio
_____Bueno
_____Malo
_____Pésimo

Pregunta 8: ¿Existe información disponible sobre los equipos de cómputo para fines de administración y toma de decisiones?

_____Si
_____No