UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Un análisis comparativo de herramientas ORM. Aplicación a un caso de estudio.

AUTOR: Suarez Gabriel Leonardo

DIRECTOR ACADÉMICO: Sebastián Dapoto

DIRECTOR PROFESIONAL: Augusto Ardissino

CARRERA: Licenciatura en Sistemas

Resumen

Los lenguajes de programación orientados a objetos son los más utilizados en los sistemas de software actuales. Sin embargo, las BD relacionales siguen siendo las más utilizadas. La función de un ORM es facilitar la conversión de los datos de un sistema que utiliza el paradigma orientado a objetos en registros de una tabla relacional y viceversa. Este trabajo tiene por objetivo realizar un relevamiento de las herramientas ORM disponibles, realizar un análisis comparativo y, finalmente, seleccionar y aplicar una de las herramientas a un caso de estudio.

Palabras Clave

Bases de Datos – Mapeo objeto-relacional – Herramientas ORM – SGBD relacional

Trabajos Realizados

Se realizó una investigación sobre las alternativas de ORM existentes en el mercado. Además, se realizó un análisis comparativo de tres herramientas de mapeo objeto-relacional existentes, determinando las ventajas y desventajas de cada una.
Se determinó la herramienta de mapeo objeto/relacional adecuada para el caso de estudio.
Se desarrolló una aplicación de gestión de información que permitió acompañar el crecimiento de un instituto.

Conclusiones

Se desarrolló una aplicación de gestión de información aplicando una de las técnicas de mapeo objeto-relacional investigadas.
La aplicación permitió acompañar el crecimiento de un instituto que brinda distintas capacitaciones para el fortalecimiento de los oficios y el establecimiento de mecanismos de autoempleo para aquellas personas que no encuentran lugar en el mercado laboral.
La aplicación presenta una mejora en el manejo de la información administrativa, como así también en la producción de la información que asiste al proceso de toma de decisiones.

Trabajos Futuros

Los objetivos a futuro se centran en dos puntos importantes. El primero de ellos es el de actualizar el sistema con el uso de nuevas técnicas de ORM. Esto involucraría, por ejemplo, la inclusión y uso del framework Spring. Por otro lado, es necesario agilizar la gestión del presentismo de alumnos, profesores y personal administrativo. Una posibilidad de realizar esto es mediante algoritmos de reconocimiento facial o lectores de huellas dactilares, reduciendo el uso de medios en formato papel.

Fecha de la presentación: AGOSTO 2021

Universidad Nacional de La Plata
Facultad de Informática

Tesina de grado
Licenciatura en Sistemas

Un análisis comparativo de herramientas ORM. Aplicación a un caso de estudio.

Autor: Gabriel L. Suarez
Director: Sebastián Dapoto

Agosto de 2021

INDICE

1 – INTRODUCCION 3
2 – CONCEPTOS Y FUNDAMENTOS GENERALES 5
2.1 – BASE DE DATOS 5
2.2 – SISTEMAS GESTORES DE BASE DE DATOS 5
2.3 – MODELO DE DATOS 6
2.3.1 – MODELO DE DATOS RELACIONAL 7
2.3.2 – MODELO DE DATOS ORIENTADO A OBJETOS 9
2.3.3 – ELECCION DE UN MODELO DE DATOS 12
3 – MAPEO OBJETO-RELACIONAL 15
3.1 – TIPOS DE MAPEADORES OBJETO/RELACIONAL 17
3.2 – MAPEO DE OBJETOS A RELACIONAL 19
3.2.1 – MAPEADO DE HERENCIA 19
3.2.2 – MAPEADO DE RELACIONES 25
3.3 – ORM DISPONIBLES 26
3.4 – ANALISIS COMPARATIVO DE LOS ORM 29
3.4.1 – ANALISIS DE LA ARQUITECTURA UTILIZADA 29
3.4.2 – ANALISIS DEL USO DE METADATOS 31
3.4.3 – ANALISIS DE MODELOS DE CONSULTA 34
3.4.4 – ANALISIS DE MECANISMO DE TRANSACCIONES 38
3.5 – HERRAMIENTAS DE PERSISTENCIA EN JAVA EN LA ACTUALIDAD 39
4 – CASO DE ESTUDIO 42
4.1 – DOMINIO DEL PROBLEMA 42
4.2 – MODELO DE DATOS 44
4.3 – TECNOLOGIA UTILIZADA PARA EL DESARROLLO DE LA APLICACIÓN 58
4.4 – DETALLE DE LOS MAPEOS REALIZADOS 58
4.5 – ESQUEMA DE LA BASE DE DATOS 64
5 – CONCLUSIONES Y OBJETIVOS A FUTURO 66
BIBLIOGRAFIA 68

1 – INTRODUCCION

En toda organización se recolecta una gran cantidad de datos a lo largo del tiempo. Estos datos son clasificados, procesados y modificados con el objetivo prioritario de mejorar el funcionamiento de la organización.

El procesamiento de grandes volúmenes de información en formato papel se torna demasiado complejo, por lo que las organizaciones tienen la necesidad de informatizar los datos y optan por la incorporación de un sistema de gestión de bases de datos.

Una base de datos (BD) se puede definir como una colección de datos interrelacionados entre sí que contienen información trascendente para cumplir el objetivo para el cual fue construida.

Un sistema de gestión de bases de datos (SGBD), incorpora al sistema un conjunto de programas para crear y mantener una base de datos. Un SGBD proporciona protección de la información almacenada contra el mal funcionamiento del hardware o software, protección sobre accesos no autorizados al sistema, acceso concurrente a los datos almacenados, entre las funciones más importantes.

Existen diferentes tipos de bases de datos. El modelo de datos relacional está basado en un conjunto de archivos denominados tablas, integradas por filas y columnas. Cada fila representa un registro del archivo y se denomina tupla, y cada columna representa un atributo del registro con un dominio de valores específicos. Este tipo de representación de una base de datos es actualmente el más utilizado, dado su potencial y simplicidad en comparación a otros modelos de datos.

Otro modelo de datos es el orientado a objetos (OO). Este modelo es una extensión del paradigma de programación OO. Las bases de datos orientadas a objetos (BDOO) incorporan características del paradigma OO, tales como: clases, comportamiento y herencia. Los objetos son instancias de clases, poseen un conjunto de variables (atributos), métodos y mensajes a los que el objeto responde. Las relaciones quedan definidas por contención lógica. Estas características permiten definir estructuras más

complejas que en el modelo relacional, y una fácil adaptación a sistemas desarrollados bajo el paradigma OO.

Sin embargo, las BD relacionales son más utilizadas que las BDOO [1]. Esto se debe principalmente a que existe una mayor cantidad y variedad de productos para BD relacionales y el modelo relacional es de fácil comprensión y aplicación.

Por otro lado, los lenguajes de programación orientados a objetos son los más utilizados en los sistemas de software actuales. Es muy frecuente que un sistema de software desarrollado en un lenguaje orientado a objetos utilice una base de datos relacional, haciendo necesario una conversión objetos/relacional.

La función de un modelo ORM (del inglés Object Relational Mapping) es facilitar la conversión de los datos de un sistema que utiliza el paradigma orientado a objetos en registros de una tabla relacional y viceversa. Al aplicar una técnica ORM se crea una capa de abstracción entre la lógica de negocio y la base de datos. Esto aporta encapsulamiento, seguridad, facilidad de mantenimiento de código, entre otras ventajas. Además, el desarrollador que aplica una técnica de ORM mantiene una mentalidad orientada a objetos al momento de programar.

Dado lo expuesto en esta sección, surge la idea de realizar un análisis sobre los ORM disponibles, detallando sus características generales, principales ventajas y desventajas. Además, se muestra la aplicación de una de estas herramientas en un caso de estudio. Se espera que el resultado del análisis realizado sirva de soporte al momento de seleccionar una herramienta de este tipo.

Esta tesina se encuadra dentro del contexto del Programa de Apoyo al Egreso de Profesionales en Actividad (PAEPA). El proyecto que se muestra como caso de estudio fue comenzado en el año 2011 en el Centro de Formación Profesional 413 (CFP 413).

2 – CONCEPTOS Y FUNDAMENTOS GENERALES

2.1 – BASE DE DATOS

La definición más clara y sencilla es la que define a la persistencia de datos como la capacidad que tienen éstos de sobrevivir al proceso que los creó. Cuando se propone el desarrollo de una aplicación, uno de los primeros requerimientos a tener en cuenta es la integración con un medio de soporte de almacenamiento de los datos que la aplicación produce a lo largo de su ciclo de vida. El medio de soporte para el almacenamiento de datos más utilizado es la base de datos.

Una base de datos (BD) es una colección de datos interrelacionados que contiene información relevante para quien la crea. A lo largo del tiempo el empleo de las bases de datos ha crecido de manera exponencial gracias a la facilidad para su utilización y los escasos recursos necesarios para contar con una, sumado al amplio crecimiento de los sistemas informáticos y a la automatización de muchos procesos manuales. Las bases de datos lograron un incremento en el rendimiento y la eficiencia tanto en el mundo empresarial como en el ámbito particular. Actualmente las bases de datos forman parte esencial de la mayoría de las empresas y organizaciones, y se han vuelto casi imprescindibles.

2.2 – SISTEMAS GESTORES DE BASE DE DATOS

Un sistema gestor de bases de datos está formado por un conjunto de datos interrelacionados y programas para acceder, estructurar y controlar los datos ofreciendo interfaces de acceso. Su objetivo principal es proporcionar operaciones para poder almacenar y recuperar los datos de manera práctica y eficiente. La gestión de los datos implica tareas tales como la definición de la estructura de la información almacenada, los mecanismos para manipularla, la definición de tipos de datos, entre otras.

Uno de los principales objetivos que tiene un SGBD es el control de la redundancia de datos. La redundancia de datos se da cuando un mismo dato se encuentra más de una

vez en la base de datos, generando pérdida de rendimiento y peligro de pérdida de la consistencia de los datos.

Los SGBD permiten asegurar la integridad de los datos almacenados. Para esto proporcionan mecanismos para que el diseñador defina reglas y restricciones entre los datos. También permiten el acceso concurrente a los datos almacenados y mecanismos de seguridad contra posibles caídas del sistema o intentos de acceso no autorizados.

Otra gran ventaja que provee el uso de un SGBD es el de la independencia de los datos almacenados, a través de diferentes niveles de abstracción. Existen tres niveles de abstracción: interno, externo y conceptual. De esta forma, es posible realizar cambios en las estructuras de datos sin afectar a las formas de acceso a los datos a nivel de aplicación. El nivel interno, el más bajo, es el que describe en detalle las estructuras de datos utilizadas para almacenar la información. En el nivel externo se representan las partes de la base de datos que son relevantes para cada aplicación o usuario diferente. El nivel conceptual corresponde a la descripción de los datos y de qué manera se relacionan entre sí. En este nivel se ve cómo se integran todas las vistas de los usuarios de la base de datos.

2.3 – MODELO DE DATOS

Un modelo de datos es una colección de herramientas conceptuales que permiten describir los datos, las relaciones entre los datos, la semántica asociada a los datos y las restricciones de consistencia existentes.

Los modelos de datos son medios para describir la realidad. A lo largo del tiempo se han propuesto diferentes variantes de modelos buscando un mayor nivel de expresividad y representación del mundo real.

Existen diferentes tipos de modelos según su nivel de abstracción. Por un lado, los modelos de alto nivel sirven para describir el dominio del problema del mundo real que se está modelando sin dar detalles específicos sobre las estructuras que se van a utilizar. Por otro lado, los modelos de bajo nivel permiten describir las estructuras de

almacenamiento y métodos de acceso que serán empleadas para tener un acceso efectivo a los datos.

Podemos definir tres componentes que integran un modelo de datos:

- Un conjunto de tipos de estructuras de datos.
- Un conjunto de reglas para operar los datos.
- Un conjunto de reglas que aseguren la integridad y consistencia de la base de datos.

2.3.1 – MODELO DE DATOS RELACIONAL

El modelo de datos relacional utiliza el concepto de relación matemática como principal elemento. Se basa en los principios del álgebra, la teoría de conjuntos y la lógica de predicados.

El modelo de datos relacional representa a la base de datos como un conjunto de relaciones o tablas. La base de datos queda representada como una estructura simple y uniforme.

Todo SGBD relacional cuenta con un lenguaje que permite la definición de la estructura de las tablas, la definición de los métodos de acceso a las tablas, la especificación de restricciones entre datos de las tablas, la manipulación de los datos almacenados, entre otros. Generalmente, este lenguaje es el denominado SQL (del inglés Structured Query Language).

SQL es un lenguaje declarativo de alto nivel basado en el álgebra y el cálculo relacionales que permite definir los esquemas de datos, permisos y restricciones de integridad, y efectuar consultas con el fin de recuperar y/o modificar de forma sencilla la información almacenada en una base de datos relacional. El lenguaje SQL permite además el uso y control de transacciones.

Entre los principales componentes que provee el lenguaje SQL, se pueden nombrar:

- Lenguaje de definición de datos. Permite la creación y modificación de los esquemas de las tablas de la base de datos, la definición de índices y métodos de acceso a los datos. A su vez, permite la especificación de restricciones de integridad y la definición de vistas, útiles para proveer modularización y seguridad al acceso de los datos almacenados.
- Lenguaje de manipulación de datos. Permite la manipulación de los datos almacenados, es decir, la inserción, eliminación, actualización y consulta de datos en la base.
- Gestor de recuperación. Es un módulo encargado de gestionar las transacciones, identificándolas y controlando la ejecución de estas. Se encarga de administrar el archivo de bitácora y posee algoritmos de recuperación ante fallos del sistema de base de datos.

Entre los SGBD relacionales más conocidos y utilizados se encuentran [2]:

- Oracle. Es multiplataforma, provee soporte de transacciones, escalabilidad, gran estabilidad y seguridad. Se destaca también su documentación a la hora de abordar problemas complejos. Dentro de sus desventajas se puede observar su complejidad, su elevado costo y su complejo sistema de configuración.
- MySQL. Es multiplataforma, tiene conectividad segura, permite replicación, indexación de campos de texto, selección de mecanismos de almacenamiento, entre otros. Dentro de sus principales ventajas se puede nombrar que es un software libre, veloz, seguro, fiable y tiene gran capacidad para la gestión de bases de datos de gran tamaño. Dentro de sus desventajas, tiene triggers limitados, en ocasiones es poco intuitivo, y hay escasa documentación para problemas específicos.
- Microsoft SQL Server. Soporta transacciones, permite trabajar en modo cliente-servidor de manera sencilla y permite adosar SQL de otros servidores. Dentro de sus desventajas está su excesivo consumo de memoria, la relación calidad/precio, y el alto aprendizaje que requiere su utilización.
- PostgreSQL. Es un SGBD relacional reconocido por su fiabilidad e integridad de datos. Es de código libre, multiplataforma y cumple con ANSI SQL. Admite

replicaciones que permiten la duplicación de bases de datos maestras en múltiples sitios. Incorpora interfaces nativas para ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python y Ruby. Provee reglas y vistas de manera simple y sencilla. Permite procedimientos almacenados, triggers y secuencias.

- DB2. Es un SGBD multiplataforma propiedad de IBM, posee una arquitectura similar a la de Oracle, y entre sus ventajas está el manejo de gran cantidad de triggers. Entre sus desventajas esta su elevado costo, es algo antiguo y los procedimientos se deben programar en otros lenguajes.
- Microsoft Access. Basado en tablas, formularios e informes. Es compatible con MS SQL, tiene gran flexibilidad a la hora de exportar e importar información dentro de los productos Microsoft y no requiere de grandes prestaciones a nivel máquina para poder ser utilizado. Como ventajas se puede destacar lo sencillo de utilizar que es, pero dentro de sus desventajas está el tamaño reducido de las bases de datos que puede gestionar, la falta de seguridad y los problemas en el acceso concurrente a los datos.
- SQLite. Es un SGBD relacional muy utilizado en aplicaciones móviles gracias a su reducido tamaño. Es compatible con ACID. Posee una única biblioteca que es necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas. Realiza operaciones de manera eficiente, es multiplataforma y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración. Implementa un gran subconjunto del SQL estándar, incluyendo subconsultas, generación de usuarios, vistas y triggers. Es de dominio público, por lo que es de uso libre y sin costo.

2.3.2 – MODELO DE DATOS ORIENTADO A OBJETOS

Las aplicaciones tradicionales presentan datos básicos y fáciles de modelar. La demanda de abordar tipos de datos más complejos, como por ejemplo el almacenamiento de imágenes, trajo como consecuencia nuevas formas de almacenamiento de datos. De la

necesidad de almacenamiento de información compleja surge el modelo de datos Orientado a Objetos (OO).

El paradigma orientado a objetos está basado en el encapsulamiento de la información y el código relacionado con cada objeto en una sola unidad. La interfaz entre el objeto y el resto del sistema se define mediante un conjunto de mensajes. Cada objeto está asociado con un conjunto de variables que contienen los datos del objeto. El objeto también posee un conjunto de mensajes a los que puede responder y así interactuar con el resto de los objetos del sistema. Un objeto se compone de una serie de métodos que contienen el código que implementa cada mensaje.

Los objetos similares que entienden los mismos mensajes, utilizan los mismos métodos y tienen las mismas variables, se agrupan formando una clase. El modelo de datos OO soporta jerarquía de clases y herencia. Una clase padre es denominada superclase y una clase hija o subclase hereda las variables, mensajes y métodos de su superclase. La herencia es una característica fundamental de este modelo de datos y promueve la reutilización de código, dado que no hace falta volver a escribir mensajes, métodos y funciones para todos los objetos que comparten variables. También existe el concepto de herencia múltiple, en donde cada clase puede tener más de una superclase.

Otra característica que provee el modelo OO es la de polimorfismo. De esta forma, es posible enviar mensajes sintácticamente iguales a objetos de distintas clases.

Cada objeto tiene un identificador único, generalmente generado automáticamente por el sistema. Esta es una gran diferencia con las bases de datos relacionales, en donde es el diseñador quien tiene que seleccionar un campo clave para que funcione como identificador.

La estructura interna de los objetos debe estar oculta al usuario, ya que no necesita conocerla para poder interactuar con el objeto. Esta característica se denomina encapsulamiento. Los objetos son concebidos como una cápsula cuyo interior está oculto y no puede ser alterado directamente desde el exterior. A la estructura interna del objeto se la denomina implementación y a la parte visible por el usuario se la denomina interfaz. La interfaz de un objeto se define mediante sus atributos y

operaciones. El encapsulamiento cuenta con ventajas tales como un elevado nivel de abstracción, transparencia ante modificaciones de implementación, la posibilidad de sustitución de objetos con idéntica interfaz y diferente implementación, entre otras.

Los SGBD Orientados a Objetos (SGBDOO) ofrecen flexibilidad y no están limitados a un lenguaje de consulta específico, ni tampoco a tipos de datos predefinidos. Estas características son claves al momento de diseñar una base de datos, tanto en estructuras complejas como en las operaciones que los objetos pueden realizar. Estas bases de datos fueron diseñadas específicamente para que interactúen directamente con lenguajes de programación que trabajan sobre el mismo paradigma.

Un SGBDOO es un sistema que almacena objetos incorporando todas las ventajas del paradigma OO. Puede tratar directamente sobre estos sin la necesidad de realizar conversiones sobre tablas o registros.

Además de las características propias del modelo OO, un SGBDOO proporciona persistencia de objetos, concurrencia, recuperación ante fallos y un lenguaje específico para realizar consultas.

Entre los SGBDOO más conocidos y utilizados se encuentran [3]:

- InterSystems Caché / IRIS. Proporciona características para el desarrollo de aplicaciones de uso intensivo de datos, incluyendo gestión avanzada de datos, interoperabilidad, procesamiento de transacciones y el análisis. También incluye interoperabilidad nativa, plataforma de análisis, combinación de escalabilidad horizontal y vertical según necesidad de los usuarios.
- Db4o. Está construido como una biblioteca (no como un servicio, como la mayoría). Una de sus características más importantes es su pequeño tamaño, lo que lo hace adecuado para ser utilizado en dispositivos móviles. Cumple con las propiedades ACID. Soporta concurrencia, replicación e indexación. Es de fácil utilización y admite muchas características básicas que se implementan de manera sencilla, reduciendo el costo de mantenimiento.

- ObjectStore. Proporciona una interfaz de lenguaje integrada a las características de almacenamiento persistente. Suministra gestión de transacciones y control de concurrencia. Brinda acceso a datos distribuidos y consultas asociativas. Está diseñado para proporcionar una interfaz para datos de forma persistente y datos de forma transitoria.
- Matisse. SGBD multiplataforma y de licencia libre. Cuenta con las operaciones básicas como control de transacciones y control de acceso. Dentro de sus principales características se puede destacar que cuenta con técnicas de segmentación de objetos para optimizar los tiempos de acceso. Posee mecanismos de duplicación de datos y de versionado de objetos. Da soporte para arquitecturas cliente-servidor.
- Objectivity/DB: es un gestor de bases de datos comercial multiplataforma producido por Objetivity Inc. Permite que las aplicaciones hagan persistentes a los objetos estándar en lenguajes como C++, Java o Python. Tiene procesos de bloqueo, transferencia remota de datos y servidor de consulta, es altamente escalable y utiliza una jerarquía de almacenamiento distribuido.
- GemStone: permite a desarrolladores escribir métodos que son almacenados y ejecutados directamente sobre la BD, provee un soporte de concurrencia robusto, tiene diversos modos de control de transacciones, seguridad a nivel de objeto, permite esquemas dinámicos y evolución de los objetos, y es altamente escalable.

2.3.3 – ELECCION DE UN MODELO DE DATOS

Al momento de seleccionar un modelo de datos a utilizar, es preciso analizar las ventajas y desventajas que provee cada modelo.

La principal ventaja del modelo relacional es su sencillez. El uso de la tabla como organización básica de los datos es intuitiva y muy familiar para todos los usuarios que han trabajado con hojas de cálculo (física o de software). Los datos se organizan de manera natural en el modelo, lo que simplifica el desarrollo y el uso de la base de datos.

Otra de las ventajas que presenta el modelo relacional es el acceso a los datos. Los usuarios pueden consultar cualquiera de las tablas, realizar combinaciones entre tablas,

realizar filtrados de manera sencilla, y seleccionar fácilmente los datos que se incluirán en el resultado.

Otro de los puntos fuertes del modelo relacional es la integridad de los datos, ya que mediante comprobaciones de validez asegura que los datos almacenados se encuentren dentro de los rangos definidos. La integridad referencial entre tablas evita que existan registros incompletos o huérfanos, lo que facilita el mantenimiento de la base, la precisión y consistencia de los datos almacenados. A su vez, la flexibilidad que el modelo presenta lo hace naturalmente escalable y extensible. Proporciona estructuras que pueden ser modificadas en el tiempo sin afectar al resto de los datos ya almacenados.

Por último, otro punto a favor de los SGBD relacionales es que son los más utilizados, existe una documentación muy completa y una gran comunidad activa para ser consultada ante dudas o inconvenientes.

Dentro de las principales ventajas del modelo orientado a objetos es que combina la programación orientada a objetos con tecnología de bases de datos proporcionando como resultado un sistema integrado. Permite el diseño, manipulación y almacenamiento de objetos complejos, que en un modelo relacional es más dificultoso modelar. En este modelo cada objeto posee una identidad unívoca que lo hace diferente al resto de los objetos, a diferencia del uso de la clave primaria en el modelo relacional. Provee encapsulamiento de datos, agrupar los objetos en clases y definir generalizaciones o jerarquías. Soporta sobrecarga, polimorfismo y extensibilidad. Este modelo proporciona persistencia de forma más transparente a la vista que el modelo relacional.

La metodología OO ofrece la flexibilidad de modelar objetos de datos sin la limitación impuesta por los tipos de datos y los lenguajes de consulta disponibles. Por lo cual, actualmente la mayoría de las aplicaciones se desarrollan utilizando un lenguaje orientado a objetos.

Sin embargo, uno de los inconvenientes de elegir una BDOO es que su uso esta poco extendido. Además, en algunas situaciones, la gran complejidad del modelo OO puede

traer problemas de rendimiento. Por otro lado, no existen muchas alternativas de SGBDOO y su comunidad es considerablemente menor que la de los SGBD relacionales.

Existen grandes diferencias entre los dos modelos analizados. Las BD relacionales tienen una utilización ampliamente mayor en comparación con las BDOO. Existe una mayor cantidad y variedad de productos que cuentan con gran soporte y el modelo relacional es de fácil comprensión y aplicación. El lenguaje SQL es muy utilizado en el ambiente de las bases de datos, por lo que la elección del modelo relacional no está acotado solamente a la elección de un modelo de datos, sino que también a un lenguaje de gestión de BD ya probado y ampliamente conocido.

Por otro lado, el paradigma OO es el más utilizado en el desarrollo de software. Una aplicación desarrollada bajo un lenguaje orientado a objetos junto a la utilización de una base de datos relacional, es una alternativa que aprovecha las ventajas de ambos paradigmas. Para poder optar por esta alternativa, es necesario disponer de una forma eficiente para realizar la persistencia de los objetos de datos.

De esta forma es que surgen las técnicas de mapeo objeto-relacional, que proporcionan una forma sencilla, automática y transparente de persistencia de datos para los casos en donde es necesario guardar objetos de datos complejos en una base de datos relacional.

3 – MAPEO OBJETO-RELACIONAL

Al desarrollar una aplicación siguiendo el paradigma orientado a objetos existe el desafío de resolver de la mejor forma posible la persistencia de los datos, debido a las diferencias que existen entre el modelo orientado a objetos y el modelo relacional.

El mapeo objeto-relacional es una técnica de programación para convertir los objetos de datos utilizados en un lenguaje de programación orientado a objetos al sistema de tipos utilizado en una base de datos relacional.

Las bases de datos relacionales solo permiten guardar tipos de datos primitivos, por lo que no es posible almacenar de forma directa los objetos de la aplicación en las tablas, sin antes realizar algún tipo de conversión. En el momento de recuperar los datos, a su vez, es necesario realizar el proceso inverso. Es entonces cuando un ORM cobra importancia, ya que se encarga de forma automática del proceso de conversión de los objetos en tuplas y viceversa. Por lo tanto, esta técnica es un intermediario entre las estructuras de datos definidas en una aplicación mediante un modelo orientado a objetos y una base de datos relacional. El modelo relacional estará centrado en el almacenamiento y manejo de los datos mientras que el modelo orientado a objetos, en el comportamiento e interacción entre objetos.

Para poder realizar una interacción entre los modelos objeto y relacional se requiere de una estrategia que posibilite realizar un mapeo del modelo orientado a objetos hacia el modelo relacional. El objetivo es que los objetos de datos definidos con un modelo orientado a objetos se conviertan en objetos persistentes en un modelo relacional. Un objeto persistente es el que puede almacenarse y recuperarse automáticamente de una base de datos. El problema es que los objetos no se pueden almacenar ni recuperar directamente de una base de datos relacional, ya que cuentan con una identidad, estado y comportamiento además de los datos. Las bases de datos relacionales almacenan únicamente los datos. Además, los datos no tienen un mapeo directo entre los tipos de datos que se manejan en el lenguaje orientado a objetos y los tipos de datos que se manejan en un SGBD relacional. Otro punto importante es que los objetos se relacionan

utilizando referencias directas, mientras que las tablas del modelo relacional se relacionan mediante claves primarias y claves foráneas. Los SGDB relacionales no proveen tampoco ningún mecanismo que resuelva el problema de la herencia de objetos, tanto para los datos como para el comportamiento.

El desarrollo de aplicaciones orientadas a objetos requiere que el programador desarrolle estrategias de mapeo complejas y eficientes para poder transformar el modelo orientado a objetos al modelo relacional, por eso es necesario que posea conocimiento sobre ambos modelos.

Las herramientas de mapeo objeto relacional proporcionan los mecanismos necesarios para integrar ambos modelos, creando capas de acceso a los datos, automatizando el acceso y generando código complementario para poder realizar accesos. El principal objetivo que persiguen estas herramientas es el manejo de la persistencia y el trabajo a nivel código de los objetos en lugar de trabajar con las estructuras de la base de datos. Las herramientas trabajan entre los datos de la base y los objetos basándose en configuraciones y ejecuciones de consultas sobre la base de datos.

Un ORM está conformado principalmente por cuatro componentes:

- Una API que posibilita la realización de las operaciones básicas de una base de datos sobre los objetos de las clases persistentes.
- Un lenguaje para poder especificar las consultas sobre las clases.
- Un archivo para especificar los mapeos de las clases.
- Una técnica para que la implementación del ORM pueda llevar a cabo búsquedas, asociaciones u otras funciones de optimización.

La utilización de este tipo de herramienta se realiza en diferentes contextos, entre los cuáles se destacan los siguientes:

- Aplicaciones cuyas entidades siguen un ciclo de vida tipo *read-modify-write*. Este ciclo de vida obtiene un dato a partir del mapeo de los datos almacenados en la base

de datos relacional, realiza modificaciones a los datos recuperados y luego lo almacena nuevamente en la base. Este es el caso de las aplicaciones web.

- Aplicaciones de tipo *read-centric*. Son aquellas aplicaciones que dedican la mayor parte del tiempo a realizar lecturas desde la base de datos.
- Aplicaciones en un contexto en el que deban interactuar con bases de datos relacionales desde un ambiente desarrollado bajo el paradigma orientado a objetos.

3.1 – TIPOS DE MAPEADORES OBJETO/RELACIONAL

Existe una gran variedad de herramientas de mapeo objeto relacional que utilizan diferentes técnicas de mapeo. Una posible clasificación de los mapeadores se basa en la perspectiva que el mapeador puede asumir. Desde este enfoque se pueden nombrar tres diferentes tipos de mapeadores [4].

Orientado a los metadatos

En este caso el desarrollador de la aplicación escribe el código a partir de definiciones en archivos XML. Luego el sistema ORM por sí mismo es el encargado de generar el código necesario para que se hagan persistentes los objetos. La base de datos puede existir previamente y es descripta por los metadatos o el mapeador puede generar la base de datos de la misma forma que lo hace con el código. En el caso de que el sistema aún no se encuentre totalmente construido es conveniente que el mapeador construya la base de datos. Si en cambio, la aplicación utiliza una base de datos legacy, el mapeador no necesita construir la base de datos, pero si escribir el código necesario para que sea compatible con el esquema existente. La ventaja que posee este modelo es la simplicidad que le provee al desarrollador. No solamente se desentiende de la persistencia de los objetos, sino que también lo hace del mecanismo interno que lleva a cabo mapear objetos a tablas. Podemos considerar como una desventaja de este método a la perdida de flexibilidad, ya que como el código es generado automáticamente por el mapeador, el desarrollador está limitado a la funcionalidad que provee el mapeador.

Orientado a la aplicación

En este caso el desarrollador de la aplicación escribe el código en el lenguaje orientado a objetos completamente, incluyendo los objetos que van a ser mapeados. El mapeador, basado en metadatos escritos por el desarrollador, escribe automáticamente el código asociado a la persistencia de los objetos a ser mapeados. El mapeador puede generar la base de datos en la cual los objetos van a persistirse o utilizar una ya existente para tales fines.

La ventaja que posee este modelo es la flexibilidad que aporta ya que permite al desarrollador escribir la lógica de las clases que van a ser luego mapeadas a través de archivos XML. Además, este modelo tiene la ventaja de focalizarse en el modelo orientado a objetos dejando de lado la lógica de persistencia. Debido a esto muchos de los sistemas ORM en la actualidad son orientados a la aplicación.

Un requerimiento de este modelo es que los mapeadores deben ser capaces de comprender el código. Esta tarea es posible realizarla de tres formas. La primera alternativa es que el mapeador sepa interpretar el código fuente. La segunda alternativa es que el mapeador pueda interpretar un código resultante de compilar el código fuente. Por último, es posible utilizar reflection[1] para acceder a la estructura de código en tiempo de ejecución. Este método resulta ser el más complejo y el que más recursos utiliza, pero a su vez es el que brinda más versatilidad.

Orientado a la base de datos

El desarrollador de la aplicación es el encargado de construir la base de datos, el mapeador examina la base de datos y deduce el modelo de clases. Esta tarea la realiza basándose en metadatos, para luego generar el código de la aplicación. A diferencia de los mapeadores orientados a la aplicación, estos mapeadores tienen la desventaja de no utilizar el mismo lenguaje y por lo tanto no proveer flexibilidad al momento de editar las clases mapeadas. Por otra parte, este modelo puede adaptarse bien a aplicaciones

[1] Reflection: es la capacidad que tiene un programa para observar y opcionalmente modificar su estructura en tiempo de ejecución. https://docs.oracle.com/javase/tutorial/reflect/index.html.

donde las estructuras de datos y las relaciones entre los tipos de datos son realmente importantes pero la lógica de negocios no lo es tanto.

3.2 – MAPEO DE OBJETOS A RELACIONAL

Al momento de comenzar a utilizar una técnica de mapeo objeto relacional es necesario tener en cuenta algunas consideraciones importantes con respecto a los atributos de los objetos de datos:

- Un atributo de un objeto puede mapearse en cero o más columnas de una tabla en el modelo relacional.
- Algunos atributos de los objetos no van a ser persistentes como pueden ser los campos calculados u otros campos propios del paradigma orientado a objetos.
- Algunos atributos de los objetos son también objetos y esto se refleja en una asociación entre dos clases que deben tener sus propios atributos mapeados, lo cual va a generar una serie de campos en las tablas del modelo relacional.

Existen dos tipos de metadatos de mapeo. La propiedad denominada *Property mapping* describe la forma de persistir un atributo de un objeto. La propiedad *Relationship mapping* describe la forma de persistir una relación entre dos o más objetos.

Otro concepto importante es el de la información oculta o *shadow information*. Esta información se compone de datos adicionales a los propios de un objeto, necesarios para que el objeto pueda persistirse. Generalmente está asociada a datos de la clave primaria (que no tiene uso en el paradigma orientado a objetos), datos adicionales de control de concurrencia y números de versiones. Estos atributos ocultos son necesarios para que la clase pueda ser persistida en el tiempo.

3.2.1 – MAPEADO DE HERENCIA

El modelo relacional no soporta de forma nativa la herencia y por lo tanto es necesario realizar un mapeo para simular la presencia de la misma. Para poder realizar el mapeo

hay que utilizar una estrategia que organice los atributos heredados al modelo de datos relacional.

A continuación, se presenta un ejemplo de esta problemática y las diferentes estrategias de solución existentes. Cada una de las estrategias es ideal para una situación en particular. Se analiza las ventajas y desventajas de cada estrategia y se proporciona una posible implementación. En la figura 1 se puede observar una jerarquía en la que se agrega una nueva clase “Ejecutivo” como especialización de la clase “Empleado”.

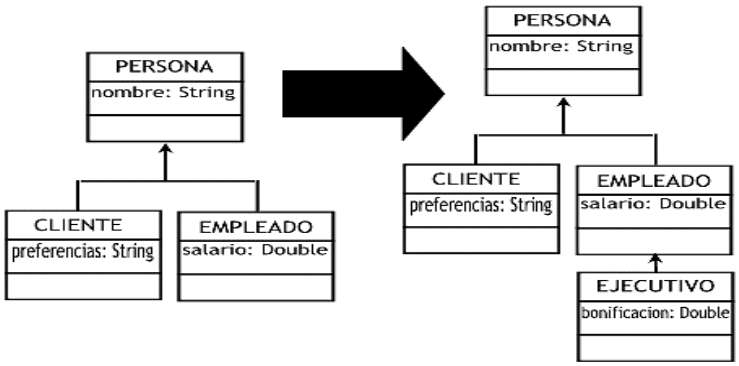

Figura 1. Ejemplo de jerarquía de clases.

Estrategia 1: mapear la jerarquía de clases en una única tabla

La estrategia consiste en mapear la jerarquía de clases en una sola tabla, es decir, se mapean todos los atributos de todas las clases de la jerarquía a una única tabla. Esta tabla generada a partir del mapeo contiene todos los campos de todas las clases. Se debe también adicionar campos necesarios en el modelo relacional como son la clave primaria para mantener relaciones entre las tablas y un campo auxiliar que se encarga de determinar de qué clase es la instancia persistida, lo cual permite reconstruir de manera eficiente el objeto desde la base de datos. Esta estrategia se puede ver aplicada en la figura 2.

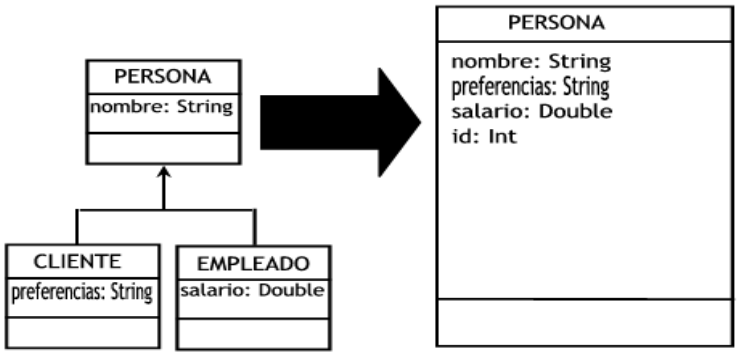

Figura 2. Estrategia de mapeo de una jerarquía de clases en una única tabla.

Si el modelo evoluciona y se agrega la clase “Ejecutivo”, es necesario agregar el campo bonificación a la tabla, permitiendo que acepte el valor nulo, ya que los clientes o empleados no definen un valor para este campo. Esto se puede observar en la figura 3.

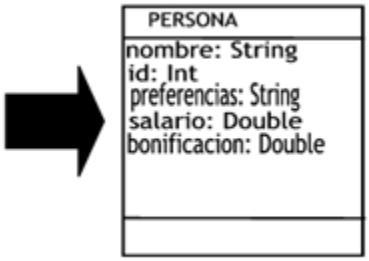

Figura 3. Incorporación de la clase Ejecutivo en la estrategia 1.

Entre las ventajas de esta estrategia se puede nombrar su simplicidad, ya que permite agregar nuevas clases de forma sencilla solamente agregando columnas a la tabla. Soporta polimorfismo, ya que mediante la consulta al valor discriminador se puede saber de qué clase se trata. Por último, este modelo provee un acceso muy eficiente a los datos sin necesidad de hacer complicadas uniones de tablas.

Entre las desventajas que esta forma de mapeo tiene, se puede observar que un cambio de jerarquía afecta a todas las clases, ya que persisten en la misma tabla. Existe además un desperdicio de espacio en la base de datos, ya que algunos campos son sólo utilizados por tuplas específicas. Si la jerarquía de clases es demasiado compleja el resultado puede ser una tabla de gran tamaño.

Esta estrategia puede ser utilizada para clases simples o jerarquías de escasa complejidad donde los tipos de jerarquía no se superponen.

Estrategia 2: mapear cada clase concreta a su propia tabla

La estrategia consiste en crear una tabla por cada clase concreta haciendo redundante todos los atributos y no mapeando clases abstractas. En la figura 4 es posible ver que la clase “Persona” es abstracta y, por lo tanto, en esta forma de mapeo no se debe tener en cuenta. Simplemente se crean las tablas “Cliente” y “Empleado” junto a la información necesaria en el modelo relacional, es decir, las claves primarias.

Al agregar la clase “Ejecutivo” se crea una nueva tabla con todos los atributos que este posee junto a su clave primaria para identificarlo unívocamente. En cualquier momento un empleado puede ser ascendido a ejecutivo y se debe tener en cuenta que este cambio requiere una copia de los campos de la tabla “Empleado” a la tabla “Ejecutivo” y posiblemente mantener la información redundante en las dos tablas para no afectar los procesos de negocio que manipulan empleados. Esto se puede observar en la figura 4.

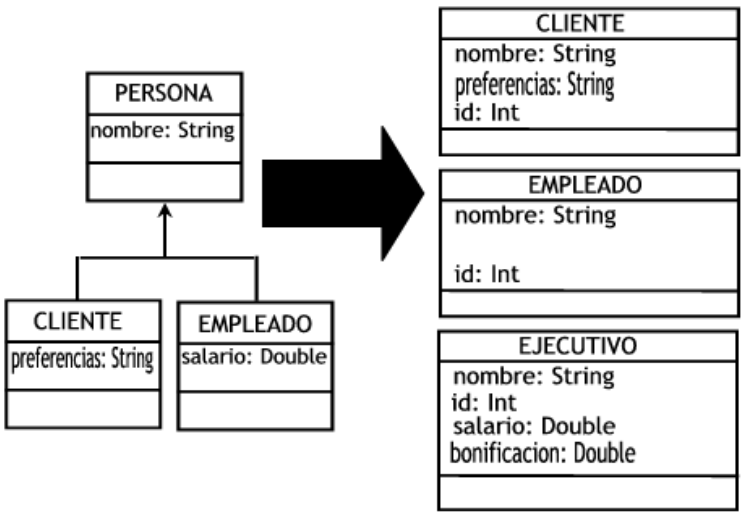

Figura 4. Estrategia de mapeo en donde cada clase concreta tiene su propia tabla.

Una de las ventajas de este tipo de mapeo es su facilidad para realizar consultas sobre una clase específica, ya que todos los datos necesarios están en una sola tabla. Esto también mejora la performance de acceso a estos datos.

Como desventaja, es una solución poco escalable, debido a que ante cualquier modificación de una clase es necesario hacer una restructuración de la tabla modificada y las tablas derivadas para poder reflejar así los cambios. La actualización de datos es compleja dado que si existen cambios de roles es necesario copiar datos y existe redundancia para mantener las reglas de negocio.

Este tipo de mapeo se puede aplicar cuando la jerarquía de clases es estable, es decir, no hay mutaciones constantes de clases o cambios profundos de roles. También puede utilizarse cuando no hay solapamiento de tipos.

Estrategia 3: mapear cada clase a su propia tabla

En esta estrategia se crea una tabla por cada clase de la herencia sin redundar atributos de las superclases. Para esto si se necesita recuperar una instancia de una determinada clase es necesario recurrir a uniones de tablas por medio de su clave primaria y así recuperar todos los datos de una determinada clase. Como se especifica, también es necesario incorporar a cada clase una clave primaria. Esta estrategia se puede observar en la figura 5.

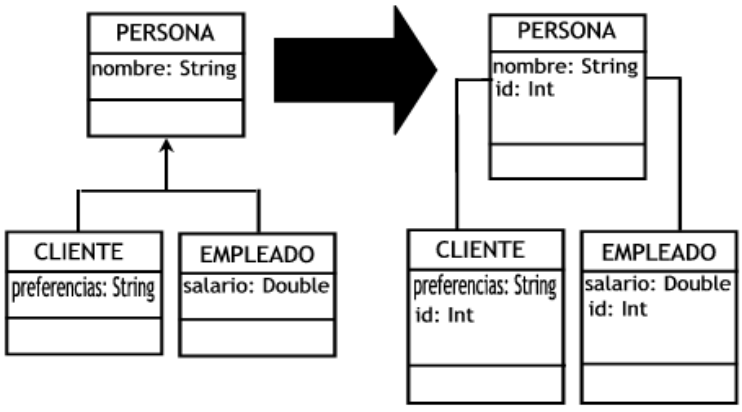

Figura 5. Estrategia de mapeo en donde cada clase tiene su propia tabla.

Al evolucionar el modelo se agrega la tabla “Ejecutivo”, siendo necesario hacer uniones entre la tabla “Persona”, “Empleado” y “Ejecutivo” para poder recuperar todos sus datos.

Esta forma de mapeo es fácil de entender, ya que se trata de un mapeo uno a uno, y soporta polimorfismo, dado que almacena los registros en la tabla correspondiente. Es fácil modificar o agregar nuevas subclases, ya que sólo implica modificar o agregar una sola tabla. Por último, el tamaño de los datos almacenados crece en proporciones directas al número de objetos persistidos.

Las desventajas de esta forma de mapeo es que existen muchas tablas en la base de datos, una por cada clase y tablas auxiliares para definir relaciones. La performance es más baja que para el resto de las formas de mapeo de datos. Esto se debe a que se necesita leer y escribir sobre varias tablas y las consultas son más complejas dada la necesidad de realizar más uniones.

Esta técnica es utilizada cuando hay solapamiento de tipos, cuando existen frecuentes cambios en las clases o cuando es necesario por algún motivo reducir al máximo la redundancia de datos.

Estrategia 4: mapear cada clase a una estructura de tablas genérica

Esta técnica de mapeo, también conocida como mapeo orientado a metadatos, no sirve solamente para mapear jerarquías sino para todas las formas de mapeo. Un esquema sencillo de esta técnica sería una tabla para mapear las clases, otra tabla para mapear los atributos de cada clase, otra tabla para guardar los valores que mantienen los atributos y una última tabla en donde se almacena la herencia de clases. Esta alternativa genérica requiere de complicadas uniones y complejas consultas para poder recuperar información persistida en las tablas, lo cual reduce sensiblemente la performance de la base de datos y del sistema en general.

Las ventajas de esta forma de mapeo es que tiene buen funcionamiento cuando el acceso a la base de datos esta encapsulado por un ORM de persistencia robusto. Además, puede soportar un amplio rango de mapeos, incluyendo mapeos de relaciones complejas. Es muy flexible y permite cambiar las formas de cómo se almacenan los objetos actualizando los metadatos almacenados en las tablas.

De todas formas, es una técnica poco utilizada, ya que puede ser difícil de implementar. Además, es posible que se necesite una aplicación extra para la administración de los metadatos y las consultas suelen ser complejas debido a la gran cantidad de uniones que se deben realizar para poder responder a las mismas.

Es una alternativa razonable para aplicaciones que manejan poca cantidad de información o aplicaciones en donde no es común realizar exhaustivas consultas a la base de datos.

3.2.2 – MAPEADO DE RELACIONES

Existen tres tipos de relaciones entre objetos que necesitan ser contempladas al momento de realizar un mapeo: la asociación, la agregación y la composición. Una asociación es una relación débil e independiente entre dos objetos. Una agregación es una relación más fuerte que una asociación, pero aun independiente. Una composición es una relación fuerte y dependiente entre dos objetos. Todas las relaciones en una base de datos relacional son bidireccionales y se pueden clasificar en base a su cardinalidad como “Uno a Uno”, “Uno a Muchos” o “Muchos a Muchos”. Dado que en las bases de datos relacionales las relaciones se mantienen mediante el uso de claves foráneas, es posible definir:

- Uno a Uno: la clave foránea esta implementada en una de las tablas.
- Uno a Muchos: la clave foránea va desde la tabla de a uno hacia la tabla de muchos.
- Muchos a Muchos: para este caso en particular es necesario crear una tabla auxiliar que sirva de soporte para contemplar la asociación.

3.3 – ORM DISPONIBLES

Es posible encontrar una gran variedad de alternativas de mapeadores objeto relacional disponibles. Existen mapeadores de licencia de código abierto que pueden ser descargados gratuitamente de internet y también productos comerciales desarrollados bajo licencias privadas. A continuación, se detallan algunos de los productos de mapeo objeto-relacional disponibles y sus principales características.

Hibernate [5]. Es uno de los ORM de código abierto no comercial más conocidos y utilizados. Su desarrollo comenzó en 2001 y originalmente fue para el lenguaje JAVA. En 2005 se desarrolló una versión para .NET bajo la denominación NHibernate. Es un sistema de ORM robusto, de alta performance y ágil al momento de realizar los mapeos. Permite desarrollar objetos persistentes incluyendo sus asociaciones, herencia, polimorfismo, composición y colecciones. Ofrece su propio lenguaje de consultas denominado HQL, aunque también permite expresar consultas en SQL nativo y Criteria basado en el lenguaje Java. Hibernate permite desarrollar clases persistentes siguiendo expresiones idiomáticas naturales orientadas a objetos. No requiere interfaces o clases base para clases persistentes y permite que cualquier clase o estructura de datos sea persistente. Admite gran cantidad de estrategias de recuperación y bloqueo ofreciendo en todo momento un rendimiento superior sobre el código JDBC nativo, tanto en términos de productividad del desarrollador como en tiempos de ejecución. Hibernate es conocido por su excelente estabilidad, configurabilidad, extensiblidad, y calidad, y es utilizado por una gran cantidad de desarrolladores en Java.

MyBatis [6]. Es una evolución del ORM open source creado en 2001 por el desarrollador Clinton Begin bajo el nombre de IBatis. Originalmente fue desarrollado para software criptográfico, pero luego cambio su dirección apuntando a las nuevas tecnologías de Internet. Inicialmente fue desarrollado para Java, pero también hay versiones para .Net y Ruby. MyBatis se basa en dos componentes principales, el Data Mapper y los Data Access. El Data Mapper, también llamado SQL Maps, es responsable de que los objetos sean persistentes mapeando objetos del modelo (JavaBeans) con sentencias SQL. Utiliza simples descriptores XML simplificando el uso de las bases de datos, es decir no mapea

objetos Java a tablas de bases de datos sino métodos a sentencias SQL. MyBatis Data Access es la capa de abstracción que oculta la persistencia de objetos en la aplicación proporcionando una API de acceso a los datos. MyBatis es un entorno de persistencia que soporta SQL, store procedures y mapeos avanzados, elimina casi todo el código JDBC, el establecimiento manual de los parámetros y la obtención de resultados. Puede configurarse con XML o anotaciones y permite mapear mapas y POJOs (Plain Old Java Objects) con registros de base de datos.

TopLink [7]. Es un ORM comercial cuyo propietario es Oracle, y es uno de los ORM más robustos que existe. Permite almacenar objetos Java en bases de datos o convertir objetos en documentos XML. Su documentación es abundante y posee una buena interfaz para el desarrollador. Dispone de una versión open source llamada TopLink Essentials. La funcionalidad principal de TopLink es proporcionada por EclipseLink, el mapeo de código abierto y el marco de persistencia. El proyecto EclipseLink está liderado por Oracle y está bajo la dirección de la Fundación Eclipse. EclipseLink implementa la API de persistencia de Java (JPA), que incluye Java Persistence Query Language (JPQL), JPA Criteria API y esquema XML para definir metadatos de mapeo relacional-objeto. EclipseLink provee extensiones para JPA como el soporte para mapeo a bases de datos no relacionales, funciones útiles como aislamiento, entidades extensibles y fuentes de metadatos externos; API Java para servicios web, anotaciones adicionales, extensiones de anotación, extensiones de JPQL, extensiones de personalización de consultas JPA y extensiones de propiedades de persistencia.

ObJect Relational Bridge (OJB) [8]. Es parte del proyecto Jakarta y es un ORM que permite persistencia transparente para objetos Java contra una base de datos relacional. OJB fue diseñado para un amplio rango de aplicaciones, como sistemas embebidos, aplicaciones RCP y arquitecturas J2EE multicapas entre otros. OJB provee una API ODMG 3.0 y también una API JDO (Java Data Objects). Puede utilizarse con JSP, Servlets, SessionBeans y provee soporte especial para Bean Managed EntityBeans (BMP). OJB utiliza un mapeo basado en XML que reside en una capa de metadatos dinámica, que se puede manipular en tiempo de ejecución a través de un simple Meta-Object-Protocol (MOP) para cambiar el comportamiento del núcleo de persistencia. OJB proporciona varias características avanzadas de objeto relacional como el almacenamiento en caché de objetos , la materialización

diferida a través de servidores proxy virtuales y la gestión de bloqueo distribuido con niveles configurables de aislamiento de transacciones, admitiendo bloqueo optimista y pesimista.

Cayenne [9]. Es un poderoso y completo ORM gratuito y open source para Java. Está certificado bajo la licencia Apache. Una de las principales distinciones de Cayenne es que provee herramientas de mapeo GUI multiplataforma que admite ingeniería inversa en esquemas RDBMS, edición de proyectos de mapeo relacional de objetos y generación de código fuente Java para los objetos persistentes. Reemplaza la codificación de sentencias SQL mediante código Java, y ofrece al desarrollador mecanismos para trabajar únicamente con objetos Java abstraídos de la base de datos. También ofrece portabilidad entre casi cualquier base de datos que tenga un controlador JDBC sin necesidad de modificar el código. No requiere conocimientos de SQL y provee almacenamiento en cache para aplicaciones que necesitan agilidad y eficiencia.

PBeans [10]. Es una capa de mapeo objeto/relacional. Está diseñada con el objetivo de ser simple de utilizar y complemente automatizado. El objetivo es ahorrar tiempo y esfuerzo para centrarse simplemente en escribir clases Java, y no preocuparse por el mantenimiento de scripts SQL, esquemas basados en XML y la generación de código. PBeans permite persistir JavaBeans con muy poca asistencia del desarrollador. Dentro de sus principales características se incluyen la creación automática de tablas y evolución de esquemas. Es un ORM basado en JavaBeans anotados (@PersistentClass). Provee persistencia transitiva, consistencia de instancia y su configuración es simple y sencilla.

SimpleORM [11]. Es un sistema de mapeo objeto-relacional con funcionalidad completa para Java. Provee una implementación simple pero efectiva de mapeos objeto-relacional por encima de JDBC a bajo costo y con bajo overhead. Proporciona una funcionalidad bien definida en un pequeño archivo sin dependencias. No es necesario configurar ningún archivo XML. SimpleORM es completamente Java puro, tiene un mínimo uso de reflection, no posee preprocesamiento y ningún byte code de post-procesamiento. Permite accesos a JDBC directo sin comprometer la integridad de la base de datos. Utiliza bloqueo optimista entre transacciones.

Kodo JDO [12]. Es una implementación robusta, de alta performance y con funcionalidad completa de Oracle de la especificación JPA y la especificación JDO para la persistencia transparente de los objetos Java. A diferencia de muchas herramientas de mapeo objeto-relacional propietarias, Kodo JDO provee acceso a las bases de datos relacionales mediante el estándar JDO, permitiendo a los desarrolladores Java utilizar la base de datos relacional existente sin necesidad de conocer SQL o ser un experto en diseño de bases de datos

relacionales. Puede ser utilizado con esquemas de bases de datos existentes o puede generar automáticamente un nuevo esquema.

3.4 – ANALISIS COMPARATIVO DE LOS ORM

Dada la cantidad de alternativas de ORM disponibles, elegir el producto de mapeo adecuado para un proyecto no resulta una tarea sencilla. A continuación, se realiza un análisis comparativo de un pequeño subconjunto de herramientas de mapeo objeto relacional. Las herramientas elegidas eran muy utilizadas en la industria del software al momento del inicio del proyecto que es caso de estudio de este trabajo [2]. Las tres herramientas que se van a analizar son: Hibernate, TopLink y MyBatis. Dichas herramientas fueron elegidas en base a que Hibernate era la más popular en la industria del software escrito en Java, TopLink era la más utilizada de las herramientas comerciales y, por último, MyBatis cuenta con una funcionalidad especial de mapeo bajo sentencias SQL, lo que la hace interesante de analizar.

El análisis de cada herramienta se inicia describiendo su arquitectura y estructura interna. Luego se examina el método de manejo de metadatos utilizado para obtener la información necesaria y lograr la persistencia. Además, se describe el modelo de consultas utilizado y finalmente se detalla como las herramientas de mapeo manejan las transacciones y la concurrencia en general.

3.4.1 – ANALISIS DE LA ARQUITECTURA UTILIZADA

Hibernate. En la arquitectura de Hibernate las clases Session, Transaction y Query, entre otras, forman la capa de persistencia. Las clases definidas por el desarrollador y las de negocio consiguen ser persistentes a través de la capa de persistencia. La capa de base de datos permanece oculta a la capa de persistencia. Hibernate interactúa con la base de datos a través de una API de base de datos como JDBC. Este tipo de arquitectura es muy común en muchas de las soluciones de ORM. Las clases de la capa de negocio interactúan directamente con las clases del ORM en la capa de persistencia, que a su vez

[2] Como se mencionó anteriormente, el proyecto comenzó en el año 2011.

utilizan la tecnología de la capa de base de datos estándar para lograr la persistencia. En el caso de Hibernate se requiere que la capa de persistencia brinde servicio a la capa de negocios a través de la utilización de metadatos. Los objetos en Hibernate siguen un ciclo de vida de persistencia. El ciclo de vida describe los estados por los que un objeto atraviesa en el transcurso de su persistencia y su recuperación desde una base de datos. Los objetos comienzan siendo transitorios, es decir no están asociados a ningún registro en la base de datos, se almacenan solamente en la memoria y se vuelven inaccesibles. Los objetos en Hibernate se vuelven persistentes cuando se asocian a un registro en la base de datos, es decir cuando contienen un valor de identificación que se hace corresponder con un valor de clave primaria en alguna tabla. Los objetos persistentes participan de transacciones y admiten la funcionalidad de revertir cambios.

TopLink. Se compone de un front-end de sesión y un back-end de acceso a datos que utilizan componentes de mapeo, consulta, almacenamiento en cache y transacciones. Las aplicaciones cliente acceden a TopLink a través de la sesión. Los componentes de consulta y transacción aprovechan la cache para minimizar la comunicación con la base de datos. El componente de acceso a datos en el back-end accede a la base de datos utilizando JDBC. TopLink da soporte a diferentes arquitecturas:

- Arquitectura de tres niveles: en esta arquitectura los clientes se conectan a un servidor de aplicaciones, que a su vez se conecta a una base de datos. Un ejemplo típico es una aplicación web, en la que muchos usuarios se conectan de manera remota a un servidor web que utiliza una base de datos para persistir la información. En una arquitectura de este estilo, TopLink se ejecuta en el servidor de aplicaciones.
- EJB sesión bean facade: es una extensión de la arquitectura de tres niveles, con beans de sesión JavaBeans que contienen el acceso a la aplicación. En esta arquitectura TopLink comparte una sesión de servidor entre beans de sesión. Cuando un bean de sesión necesita una sesión de TopLink, obtiene una sesión del cliente de la sesión del servidor compartido. Las transacciones se encapsulan en beans de sesión.
- EJB 3.0 con JPA: esta arquitectura incluye JPA y un marco estandarizado para la persistencia en Java. TopLink proporciona soporte para desarrollar aplicaciones a través de JPA.

- Web Services: en este tipo de aplicaciones se encapsula la lógica de negocio en un servicio web. TopLink puede asignar el modelo de objeto a un esquema XML para usarlo con el servicio web.
- Dos niveles: conceptualmente la arquitectura de dos niveles es la más simple, pero rara vez se utiliza debido a las limitaciones de escalabilidad. En el modelo de dos niveles, los clientes aceden a la base de datos directamente. En consecuencia, cada aplicación requiere su propia sesión.

MyBatis. Posee una capa de persistencia que interactúa entre la aplicación y una base de datos. En lugar de vincular directamente clases a tablas, MyBatis asigna la entrada y la salida de la base de datos a entidades de aplicación, agregando una capa de indirección entre la aplicación y la base de datos. Este mapeo se describe mediante consultas SQL proporcionadas por el desarrollador. MyBatis no intenta ocultar los SQL de los desarrolladores, como las soluciones ORM tradicionales, sino que les proporciona control total. Debido a este flexible acoplamiento, MyBatis acomoda los sistemas en que el diseño de la aplicación y el diseño de la base de datos pueden ser totalmente incompatibles, para lo cual presenta la ventaja al momento de tratar con bases de datos existentes y compartidas, así como bases de datos cuyo diseño va mutando en el tiempo.

3.4.2 – ANALISIS DEL USO DE METADATOS

Hibernate. Para persistir objetos en Hibernate los desarrolladores escriben códigos de persistencia en archivos XML. El formato de metadatos utilizado para persistir objetos se caracteriza por su legibilidad y facilidad en su uso. El siguiente ejemplo proporciona el formato de metadatos de Hibernate:

```
<?xmlversion="1.0"?>
<!DOCTYPEhibernate-mappingPUBLIC"-//Hibernate/HibernateMappingDTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
    <classname="org.hibernate.shopping.model.Automovil"`table="Automovil">
        <id name="id"column="AutomovilId" type="long">
            <generator class="native"/>
```

```
            </id>
            <property name="marcaAutomovil" column="Marca" type="string"/>
        </class>
    </hibernate-mapping>
```

Hibernate proporciona una definición de tipo documento, los metadatos son escritos en documentos XML en un formato definido específicamente para realizar el mapeo. Esto garantiza que el formato en el que está escrito el mapeo esté libre de errores. Cada documento de mapeo tiene un elemento de mapeo RootHibernate. Las clases en Hibernate se especifican por elementos de clase, que son elementos secundarios del elemento de mapeo. Como una convención cada clase está definida en su propio documento de metadatos. Los desarrolladores especifican el nombre de la clase Java y el nombre de la tabla correspondiente. El elemento de clase tiene propiedades secundarias que especifican los datos persistentes de la clase, y los nombres de sus columnas correspondientes en la base de datos. Como se ve en el ejemplo anterior, el elemento ID se utiliza para definir la clave primaria. En el caso de que no se especifique ningún nombre para la columna, Hibernate identifica la propiedad con la columna de la base de datos con el mismo nombre, por lo que simplemente podemos definir <property name="name"/>. Otra característica que posee Hibernate es la posibilidad de manipular los metadatos de forma dinámica, en tiempo de ejecución.

TopLink. Para definir los documentos de persistencia provee herramientas gráficas que permiten realizar los mapeos. El denominado TopLink Workbench, permite establecer asignaciones entre objetos y tablas sin escribir documentos con metadatos o códigos necesarios para referenciar a una API de persistencia. TopLink también permite realizar una codificación manual. El documento de metadatos más importante se denomina Project.xml, y está compuesto por todos los mapeos y consultas. TopLink utiliza también documentos XML para metadatos y datos relacionados con la persistencia, como los datos de sesiones.

TopLink Workbench es un ambiente flexible que permite varios enfoques para establecer el mapeo. Sin embargo, la lógica que se aplica generalmente es que los

desarrolladores crean las clases y las tablas y luego utilicen TopLink Workbench solamente para establecer las asignaciones correspondientes entre ellas.
Un mapeo asocia un solo miembro de datos de un objeto con su representación en la base de datos y define la conversión entre estas dos.

MyBatis. Los metadatos más significativos se componen por los descriptores de mapeo de SQL. Estos descriptores son documentos que contienen los SQL para comprender las relaciones que existen entre la aplicación y la base de datos. El siguiente ejemplo muestra un descriptor de mapeo SQL de MyBatis.

```
<select
      id="getAutomovil"
            parameterClass="int"
            resultClass="Automovil">
      SELECT Id as id, Marca as marca, Modelo as modelo, NumeroPatente as numeroPatente FROM
      Automovil WHERE Id = #id#
</select>
```

MyBatis ejecuta este código SQL en la base de datos en el transcurso de las operaciones de persistencia. El desarrollador es el que escribe el código SQL que finalmente es el código de mapeo. El siguiente es un ejemplo del código necesario para recuperar un registro de la base de datos.

```
Automovil automovil - (Automovil)sqlMap.queryForObject("getAutomovil", newInteger(2));
```

Los documentos de mapeo permiten aplicar también elementos como <insert>, <update>, <delete> y <procedure>. Otros elementos disponibles son <sql> e <include> que permiten a los desarrolladores crear consultas complejas a partir de pequeños bloques de SQL reutilizables, como se muestra en el siguiente ejemplo.

```
<sql id="select-min-precio"> select min(Precio) as valor from Automovil </sql>
<sql id="select-max-precio"> select max(Precio) as valor from Automovil </sql>
<sql id="where-automoviles-recientes"> <![CDATA[ WHERE fecha > #value:fecha# ]]> </sql>
<select id="getMinPrecioDeAutomovilesRecientes" resultClass="java.math.BigDecimal">
      <include refid="select-min-precio"/>
      <include refid="where-automoviles-recientes"/>
</select>
```

```
<select id="getMaxPriecioDeAutomovilesRecientres" resultClass="java.math.BigDecimal">
    <include refid="select-max-precio"/>
    <include refid="where-automovies-recientes"/>
</select>
```

3.4.3 – ANALISIS DE MODELOS DE CONSULTA

Hibernate. Permite realizar consultas de tres formas diferentes, mediante HQL (lenguaje nativo de Hibernate), Criteria y Query By Example (QBE). HQL es un lenguaje de consulta similar a SQL que Hibernate define para permitir que la aplicación realice consultas haciendo referencia a las estructuras por los nombres en la aplicación y no por los nombres definidos en la base de datos. Cuando la aplicación crea una consulta HQL se llama al método createQuery de la sesión y este devuelve una instancia de la clase Query, como se puede ver en la siguiente sentencia:

```
Query quería = session.createQuery(hqlQueryString);
```

El tipo más simple de consulta HQL es aquella que recupera todos los registros de una tabla dada. Además, admite consultas con cláusulas de restricción. En las siguientes sentencias se puede observar un ejemplo de cada caso.

```
Query query = session.createQuery("fromAutomovil");
From Automovil a where a.marca ='Chevrolet’;
```

Se pueden realizar todo tipo de comparaciones y operaciones lógicas y soporta también conjuntos de resultados ordenados: una vez ejecutada la consulta se obtiene el resultado como una lista, como se puede observar en el siguiente ejemplo.

```
List result = query.list();
```

HQL permite utilizar parámetros posicionales o con nombre, como se puede ver en el siguiente ejemplo. Esto es útil ya que, además de proteger el sistema contra ataques de inyección SQL, los parámetros con nombre permiten mejorar el rendimiento ya que prevén que la base de datos precompile la consulta.

```
String queryString –
  "from Automovil a where a.marca=:searchString";
Query query = session.createQuery(queryString).
  setString("searchString","Chevrolet");
List result = query.list();
```

Las consultas de Hibernate se pueden colocar en los metadatos como consultas con un nombre dado. En la aplicación, es posible recuperar estas consultas llamando al método getNamedQuery() de una sesión. Realizar la tarea de recuperar consultas es una mejor solución que dispersar las consultas por todo el código de la aplicación.

```
<query name="findRecientesAutomovilesPorMarca">
<![CDATA[ from Automovil a
                where a.marca like :marca and a.fecha >= :fecha ]]>
</query>

Query query - session
.getNamedQuery("findRecientesAutomovilesPorMarca ")
.setString("marca",marca)
.setDate("fecha", fecha);
```

El segundo método de consulta que Hibernate provee es Criteria. Para utilizar este método de consultas, la aplicación crea un método Criteria basado en una clase y establece restricciones en él. Por ejemplo:

```
Criteria criteria = session.createCriteria(Automovil.class);
Criterion marca = Expression.eq("marca","Chevrolet");
criteria.add(marcaChevrolet);
List result = criteria.list();
```

El primer paso es crear una entidad Root basada en la clase apropiada, en este ejemplo es Automovil. En este caso, el único Criterion es la marca del automovil, o sea “Chevrolet”. Luego se ejecuta el Query llamando a callinglist() o a uniqueresult().
El tercer método de consulta en Hibernate es QBE. En este caso, la aplicación muestra un ejemplo del tipo de resultado que le gustaría recibir. Es decir, proporciona una instancia real del tipo que está consultando para recibir resultados que coincidan con este tipo. El siguiente ejemplo consulta por vehículos con marca “Peugeot”:

```
Automovil automovil =newAutomovil();
automovil.setMarca("Peugeot");
Criteria criteria = session.createCriteria(Automovil.class);
criteria.add(Example.create(automovil));
List result = criteria.list();
```

TopLink. Admite nueve tipos de consultas diferente, pero los más básicos son los cuatro que se detallan a continuación. Session queries son consultas de sesión. Este tipo de consultas es construida y ejecutada por un objeto de sesión y es capaz de realizar las acciones de persistencia más básicas de un objeto. Database queries son consultas a la base de datos. Se utiliza una instancia de la clase DataBaseQuery construida exclusivamente para representar una consulta. Puede realizar cualquier acción de persistencia en objetos o datos. Named queries son consultas con nombre. Se utiliza una instancia de DataBaseQuery almacenada por nombre en un objeto de sesión o descriptor. Es construido una única vez, pero puede ejecutarse repetidas veces. Call queries son consultas de llamadas y se usan para ejecutar código SQL y store procedures. Los desarrolladores cuentan con la posibilidad de escribir las consultas a través de TopLink Workbench o lo pueden realizar mediante la TopLinkAPI. Los objetos de sesión admiten operaciones de consulta simples, (leer, escribir o eliminar objetos). Por ejemplo, para encontrar todos los automóviles que existen con un peso superior a 300 kg se puede escribir las siguientes sentencias:

```
Vector results = session.readAllObjects(c.class,
newExpressionBuilder.get("automovil").greaterThan(300));
```

Las consultas a la base de datos son flexibles. El uso de una consulta de este tipo para encontrar los automóviles cuyo peso supera los 300 kg, requiere algunas líneas más de código.

```
ReadAllQuery query =newReadAllQuery(Automovil.class);
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCriteria(builder.get("automovil").greaterThan(300));
Vector results = (Vector) session.executeQuery(query);
```

El sistema de consultas QBE es muy flexible. Admite comparaciones, operaciones sobre strings e incluso búsquedas por palabras clave. La siguiente consulta encuentra los automóviles cuyo peso supera los 300 kg:

```
Automovil automovil =new Automovil();
automovil.setPesoTotal(300);
ReadObjectQuery query =newReadAllQuery(Automovil.class);
query.setExampleObject(automovil);
Vector results = (Vector) session.executeQuery(query);
```

Las consultas a base de datos tienen muchas características y opciones. Las consultas con nombre son como consultas a base de datos excepto que están asociadas a un nombre y predefinidas anteriormente. La sintaxis más simple es la siguiente:

```
result = session.executeQuery("buscarLargoDeAutomovil");
```

Finalmente, las consultas de llamada son los medios más flexibles de consulta en TopLink. Permiten ejecutar sentencias SQL directamente en la base de datos. Para utilizar este tipo de consultas se debe crear un SQLCall, un StoreProcedureCall o un StoreFunctionCallObject y pasarlo a la sesión.

MyBatis. Las consultas se definen mediante SELECTSQL en el interior de elementos <select> contenidas en documentos de mapeo. Como consecuencia MyBatis permite a los desarrolladores especificar y ejecutar esencialmente cualquier instrucción SQL que el SGBD acepte. Un ejemplo puede ser el siguiente:

```
Automovil automovil = (Automovil) sqlMap.queryForObject("getAutomovil", id);
```

Cuando se ejecuta esta línea de código, la API de MyBatis busca la instrucción mapeada getAutomovil y trasforma los parámetros en línea de parámetros de instrucción produciendo un SQL como el siguiente:

```
SELECT Id as id, Marca as marca, Modelo as modelo,
NumeroPatente as numeroPatente
FROM Automovil where Id = ?
```

MyBatis envía esta declaración a la API JDBC, establece el valor del parámetro, ejecuta la instrucción, y luego asigna la fila resultante a un objeto y lo devuelve.

3.4.4 – ANALISIS DE MECANISMO DE TRANSACCIONES

Hibernate. Admite tanto un mecanismo de bloqueo pesimista como optimista. La interfaz de transacciones proporciona métodos para declarar los límites de una transacción. En el siguiente ejemplo, una transacción comienza con Session.beginTransaction(), realiza algunas tareas de persistencia y luego cierra la transacción con commit() para confirmar la transacción, o mediante un rollback() para revertirla si llegara a producirse algún error.

```
Session session = getSessionFactory().openSession();
Transaction tx =null;
try{  tx = session.beginTransaction();
        Automovil peugeot = (Automovil) session.get(Automovil.class, 1);
        peugeot.cambiarPeso(1100.00);
        Automovil fiat = (Automovil) session.get(Automovil.class, 2);
        fiat.cambiarPeso(1000.00);
        tx.commit();}
catch(Exception e) {
        if(tx !=null) tx.rollback();
        throw e;}
finally{
        session.close();
}
```

TopLink. Las transacciones están representadas por objetos unit work. Es posible adquirir una unit work llamando al método acquireUnitOfWork() de un objeto de sesión. La unidad permanece utilizable hasta que se llama al método commit() o al método release(). Internamente la unit realiza cambios en los duplicados de objetos en una cache interna de la unit work. Cuando se realiza el commit sin recibir notificación de algún problema, los cambios se replican en la base de datos (y en la cache de la sesión). UnitOfWork deriva de la sesión y por lo tanto admite los mismos métodos que la sesión. TopLink admite bloqueos de tipo optimista y pesimista. Con un bloqueo optimista, todos

los procesos tienen acceso a la lectura de datos. Cuando un proceso intenta efectuar un cambio, la capa de persistencia verifica que los datos no se hayan modificado desde la última vez el proceso leyó los datos. Con el bloqueo pesimista, el primer usuario que accede a los datos para actualizarlos, los bloquea hasta finalizar la actualización de los mismos. Una característica excepcional de TopLink es el control de transacciones externas. En una arquitectura de aplicación con muchos usuarios que se conectan a un servidor de aplicaciones puede proporcionar un servicio de transacciones que gestione las transacciones en un nivel total.

MyBatis. Toda la persistencia de MyBatis se enmarca dentro del alcance de una transacción. Las transacciones no necesitan ser demarcadas explícitamente, sino que las ejecuciones de sentencias se envuelven automáticamente en transacciones. Las transacciones automáticas son transparentes para los usuarios. Otra alternativa es que las transacciones sean controladas por la aplicación a través de SQLMapClient, como se ve en el siguiente ejemplo.

```
try{
        sqlMapClient.startTransaction();
        sqlMapClient.commitTransaction();}
finally{
        sqlMapClient.endTransaction();}
```

MyBatis también proporciona soporte para transacciones globales, transacciones que pueden involucrar múltiples bases de datos o múltiples aplicaciones. También hay soporte para el manejo de transacciones personalizadas, ya sea definiendo un manejador de transacciones o asumiendo el control del objeto JDBCConection que MyBatis utiliza.

3.5 – HERRAMIENTAS DE PERSISTENCIA EN JAVA EN LA ACTUALIDAD

Desde el momento en que se realizó el desarrollo de software que es caso de estudio de esta tesina a la actualidad, han surgido nuevas herramientas y técnicas de persistencia.

Como se verá en la sección 4.3, la aplicación fue desarrollada en Java y se utilizó la herramienta de mapeo objeto-relacional Hibernate.

Si bien Hibernate continúa siendo una de las herramientas de mapeo más utilizadas, existen nuevos frameworks para Java que también son muy usados. A continuación, se detallan algunos de ellos.

jOOQ (Object oriented Quering) [13] es un ORM orientado a traducir el modelo relacional en lugar de traducir el modelo de objetos como ocurre en la mayoría de los ORM. jOOQ permite escribir objetos relacionales en la base de datos utilizado SQL y luego genera el código Java para mapear los mismos. jOOQ se basa en el poder que tiene SQL, que hace posible desarrollar la base de datos junto con su esquema sin tener que preocuparse por como lo va a manejar Java. jOOQ proporciona una interfaz simple para los casos en los que no se requiere utilizar todas las complejidades y herramientas disponibles en JPA. Basado en la forma en que está diseñado jOOQ, resulta muy fácil desarrollar aplicaciones Java sobre una base de datos existente. jOOQ ayuda a generar todas las clases y objetos modelados automáticamente. En jOOQ la seguridad de tipos garantiza el reconocimiento de errores en tiempo de compilación en lugar de en tiempo de ejecución (una de las principales complicaciones en JDBC). Posee una documentación completa sobre el conjunto de funciones disponibles que se muestran en su sitio web.

ActiveJDBC [14] es un ORM liviano basado en las características de ActiveRecord uno de los principales ORM del lenguaje Ruby. Está centrado en simplificar la interacción con las bases de datos eliminando la capa adicional que proporcionan los ORM típicos y centra su uso en SQL en lugar de crear nuevos lenguajes de consulta. Una de sus principales características es la inferencia de los parámetros del esquema de una base de datos, eliminando de esta forma la necesidad de mapear entidades a tablas. En ActiveJDBC no existen las sesiones, tampoco los administradores de persistencia ni lenguajes de consulta propios. La biblioteca provista es simple y de acotado tamaño reduciendo el mantenimiento de las bases de datos. Su principio de diseño se basa en la inferencia de metadatos en la base de datos. El modelado es simple y está basado en POJOs. No requiere objetos de tipo DAO (Data Access Object) ni DTO (Data Transfer Object).

Spring Data [15] es uno de los frameworks de persistencia y acceso a los datos más utilizados en la actualidad. Ofrece un modelo de acceso a los datos, configuración automática y manejo de beans. El contenedor de Spring crea los objetos, los une, los configura y administra completamente su ciclo de vida. Spring Data brinda la posibilidad de realizar consultas con el solo hecho de mencionar los métodos de acuerdo al criterio de Spring Data. Tiene un gran soporte para realización de auditorías, posibilita la modificación de repositorios ingresando código personalizado y está integrado con los controladores creados bajo el estándar Spring MVC.

Los módulos principales que el framework contiene son:

- Spring Data Commons: módulo que contiene los conceptos básicos de Spring y el código base que soporta cada módulo de Spring Data.
- Spring Data JDBC: módulo que da soporte a la API de conectividad a las bases de datos.
- Spring Data JPA: módulo de gestión de acceso a los datos y de soporte de persistencia.
- Spring Data Key Value: repositorios basados en mapas para la creación de un módulo Spring Data para almacenar datos en forma clave – valor.

4 – CASO DE ESTUDIO

En esta sección se presenta el caso de estudio particular en donde fueron aplicadas las técnicas vistas en este trabajo. Inicialmente se detalla el dominio del problema y se describe el modelo de datos logrado. Luego, se especifica la tecnología utilizada para el desarrollo de la aplicación y el diseño de los mapeos de datos.

4.1 – DOMINIO DEL PROBLEMA

El espacio laboral en donde se gestionó el desarrollo de la aplicación es un ámbito educacional que apunta a fortalecer la inserción laboral a través del dictado de cursos formativos para distintas áreas en donde los alumnos se van a desempeñar en un futuro. Los alumnos desarrollan distintas técnicas prácticas las cuales van ser directamente aplicadas, una vez egresados, en su entorno laboral.

La idea surge en el año 2007 en donde se detectaba una fuerte escases de trabajo en la comunidad y una pérdida progresiva de los oficios. A partir de esto, fue necesario establecer mecanismos de autoempleo, para satisfacer las necesidades de las personas que se encontraban en un determinado rango de edad (entre los 30 y 70 años), no se adaptaban a las nuevas necesidades que el mercado laboral exigía, y por lo cual, no lograban encontrar un trabajo estable.

Viendo estas necesidades nace la posibilidad de generar un proyecto educacional que, a través de cursos formativos, pudiera capacitar a los alumnos y generar de esta manera una forma de inserción al mercado laboral a través del autoempleo.

Se seleccionaron las capacitaciones que ofrecían una mayor salida laboral de manera rápida y de forma personal, sin depender de las exigencias y/o necesidades que tienen los diferentes ámbitos empresariales. De esta manera, los alumnos que cumplían con los requisitos de una capacitación cursada, tenían la posibilidad de generar autoempleo, sin la necesidad de continuar presentándose infructuosamente en diferentes entrevistas laborales.

El proyecto fue sumamente exitoso y la institución contaba cada vez con más alumnos inscriptos, que elegían diferentes ofertas formativas viendo la posibilidad de insertarse

en el mercado laboral de manera autónoma. Así surge la necesidad de generar informes y manejar información sensible de cada uno de los cursos, alumnos, profesores, entre otros. Esta información debía ser generada con rapidez, exactitud y libre de errores.

Con el paso del tiempo y la gran cantidad de alumnos inscriptos, el volumen de información en formato papel se hizo muy complicado de manipular, por lo que fue necesario buscar alternativas de digitalización de los datos a medios electrónicos. Esto permitiría, a su vez, agilizar el análisis de datos estadísticos del alumnado y de los cursos formativos.

Desde el Ministerio de Educación no se contemplaba el uso de ningún medio electrónico o aplicación para realizar el manejo de la información en las diferentes instituciones. Por esto, se procedió a realizar una búsqueda de distintos sistemas de software que contemplaran las características básicas necesarias para almacenar la información que se manejaba. Las aplicaciones que fueron probadas eran bastante simples y antiguas [3]. Estas aplicaciones cumplían en parte con el objetivo de un acceso más eficiente a la información, pero sin embargo no tenían una conexión entre ellas, ya que eran de distintos autores y utilizaban distintas bases de datos, por lo que no era posible recuperar información completa y detallada. Esto era un problema, ya que, por ejemplo, para completar un informe se debía recuperar información de forma manual de las distintas aplicaciones.

Por todo esto se decidió finalmente comenzar con el diseño de una aplicación propia, y de esa forma lograr integrar la completitud de los datos que manejaba la institución. Un requerimiento importante para la nueva aplicación era la funcionalidad de generar los diferentes informes de forma automática, ya que la mayoría de estos informes y certificaciones exigidas por el Ministerio de Educación aún se solicitan en formato papel.

A continuación, se detalla el modelo de datos que surge del relevamiento realizado. El modelo incluye todos los datos relacionados con los cursos y los alumnos, para posibilitar el análisis de datos estadísticos anuales y facilitar el proceso de toma de decisiones.

[3] No fue posible encontrar una mayor información sobre estas aplicaciones, dado que las computadoras en las que estaban instaladas se perdieron en la inundación de la ciudad de La Plata en el año 2013.

4.2 – MODELO DE DATOS

En esta sección se detalla el modelo de datos generado en el proceso de relevamiento de la información manejada en la institución. Este modelo de datos ha sido modificado en varias oportunidades, dado que año a año se actualizan los datos e informes exigidos por el Ministerio de Educación. En las figuras 6 y 7 es posible observar el diagrama de clases, y a continuación se detalla una breve descripción de cada clase y los datos que contienen.

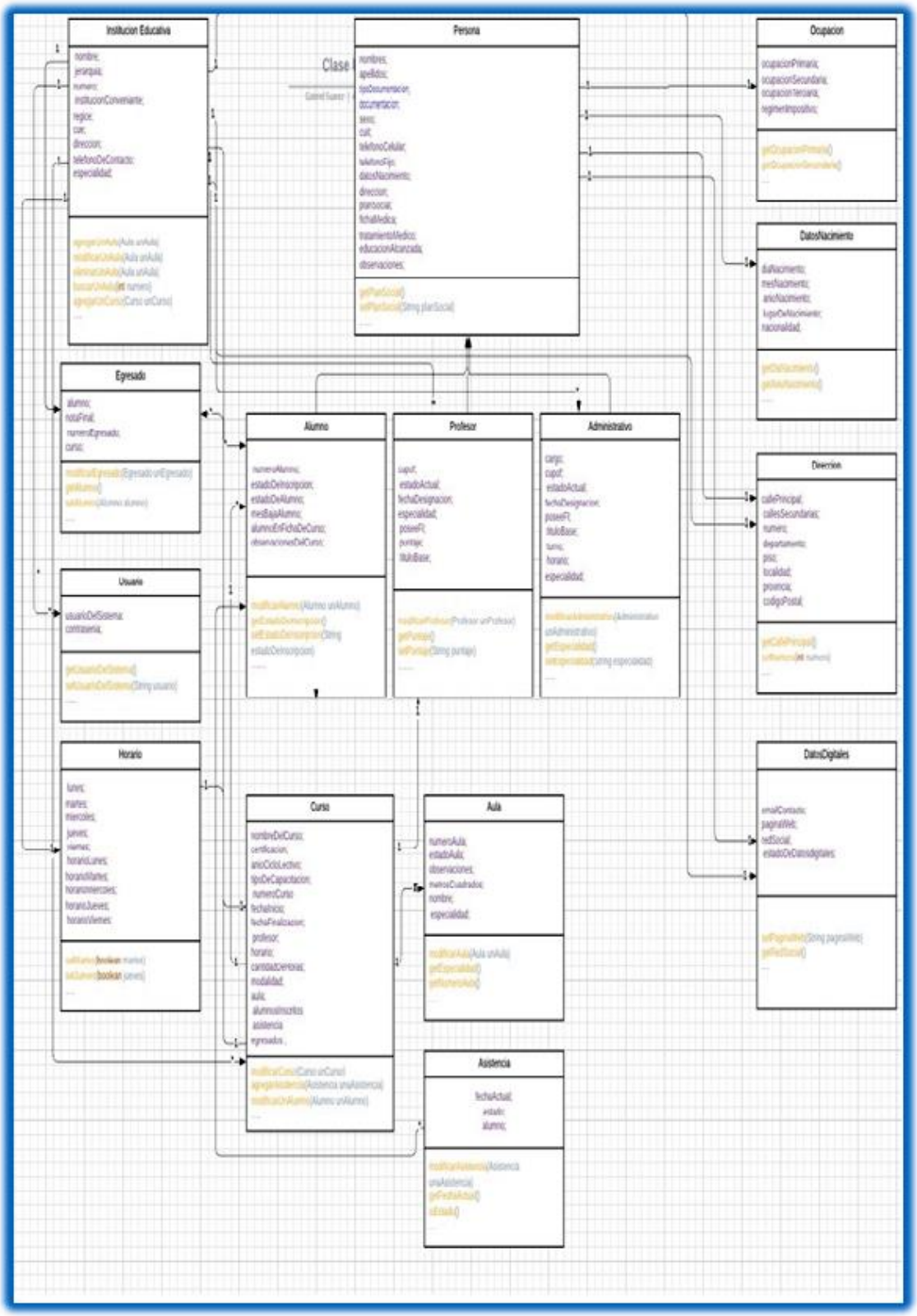

Figura 6. Diagrama de clases (parte 1).

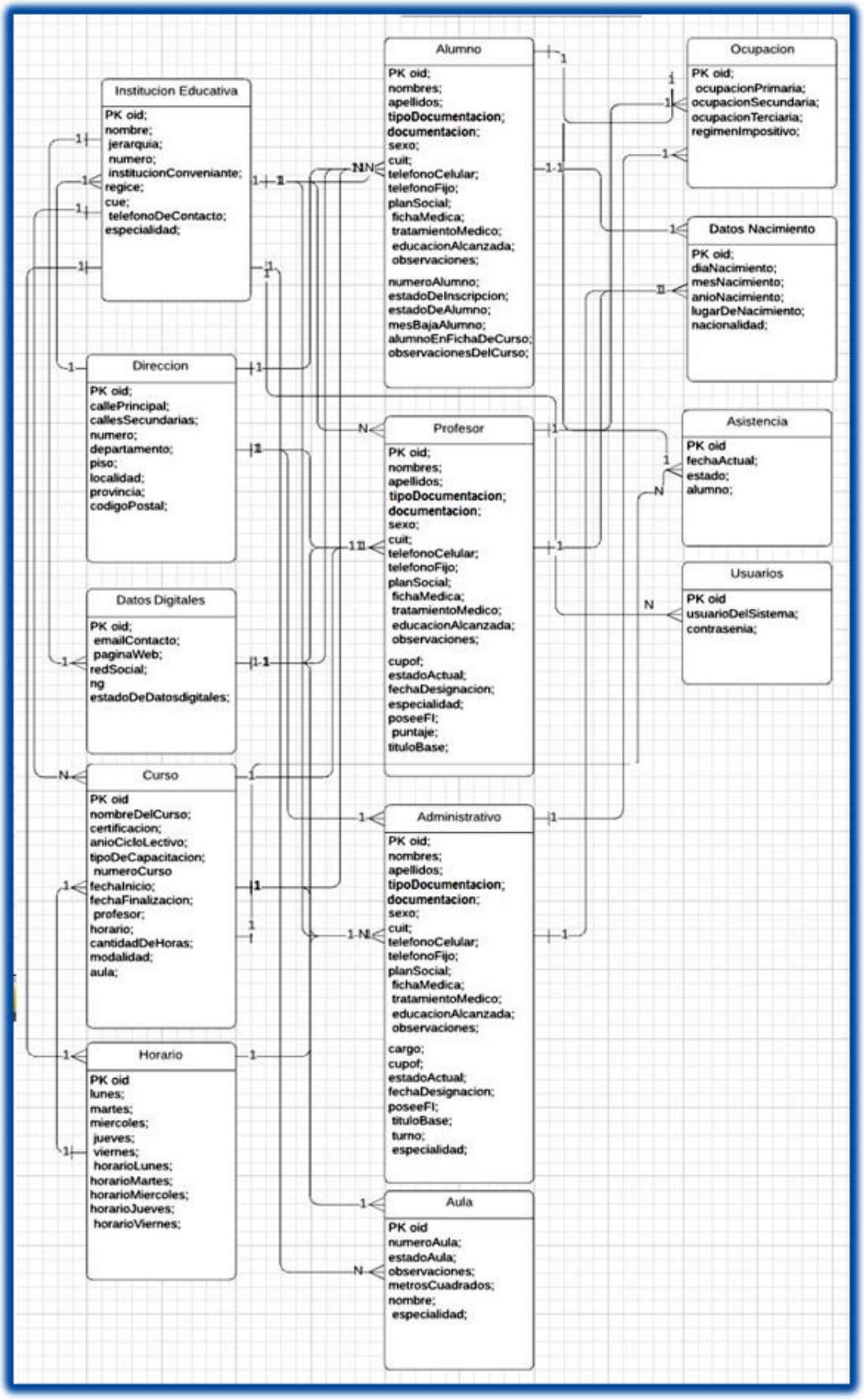

Figura 7. Diagrama de clases (parte 2).

Clase CURSO. La información que se maneja de los cursos es la siguiente.

NOMBRE DE CURSO. Es el nombre con el cual los alumnos conocen y pueden optar por elegir realizar una capacitación. El nombre del curso generalmente se condice con el nombre formulado en la nómina de cursos, pero en algunos casos es modificado para ser autodescriptivo y simplificar la interpretación de los posibles alumnos. Es importante destacar que algo tan simple como un nombre es un factor determinante para que las personas elijan o no una determinada capacitación. Un ejemplo muy claro de esto es el nombre del curso de “Instalador Sanitarista”; muchas de los posibles alumnos, al realizar una lectura de un folleto o un análisis de las ofertas formativas sobre alguna red social, no comprenden de que se trata y esto provoca múltiples preguntas. En este caso se reemplaza el nombre del curso por el de “Plomería”, término más conocido y de fácil interpretación. Esto reduce exponencialmente las preguntas sobre de que se trata el curso.

CERTIFICACION. En este caso, cada una de las certificaciones debe respetar la nómina de cursos y no puede ser modificada. Cada certificación dispone las competencias del alumno egresado y para algunos casos también dispone las competencias alcanzadas por los egresados respecto a los entes en donde se deben matricular para poder ejercer de forma legal.

AÑO CICLO LECTIVO. Es el año en el cual fue dictado el curso. Es un dato importante ya que en determinados cruces de información no se manejan las fechas de inicio y finalización de cada curso, sino que se utiliza solamente el nombre del curso, la certificación, el año en que se dictó y el número.

TIPO DE CAPACITACION. Está relacionado con el tipo de curso que se va a dictar. Los más utilizados en la actualidad son, por ejemplo, los cursos de capacitación laboral, los cursos de formación y los cursos intensivos de simple jornada. El tipo de curso determina el tiempo de duración y la planificación sobre la cual están basados los cursos.

NUMERO DE CURSO. Identifica unívocamente cada una de las capacitaciones dictadas en la institución. Utilizando este número se puede determinar, por ejemplo, el profesor que estuvo a cargo del curso y los alumnos que lo realizaron.

FECHA DE INICIO. Se corresponde con el inicio de la capacitación en el año lectivo, es de conocimiento público y es la fecha en la que deben presentarse el profesor a cargo y los alumnos inscriptos a la capacitación.

FECHA DE FINALIZACION. Se corresponde con la fecha en la cual llega el curso a su fin y determina, no solamente la finalización física del curso, sino también la finalización de los actos administrativos relacionados con dicho curso. Para esta fecha los alumnos deben haber aprobado todas las prácticas relacionadas a la capacitación correspondiente.

PROFESOR. Determina la persona a cargo del dictado de la capacitación. Es el encargado de llevar a cabo la planificación del curso y manejar su evolución. El profesor también es el responsable de indicar los cursos transversales necesarios para una formación integral de los asistentes a una capacitación.

HORARIO. Determina los días y horas en los que se dicta la capacitación. La clase Horario se va a desarrollar con más profundidad más adelante en el texto.

CANTIDAD DE HORAS. Dispone el número de horas cátedra totales que son necesarias para cumplimentar con la capacitación. Este valor está establecido en la nómina de cada curso, y depende de la necesidad y complejidad de cada una de las capacitaciones.

MODALIDAD. Dictamina la duración en meses de cada una de las capacitaciones. Puede ser mensual, bimestral, trimestral, semestral o anual, dependiendo de la complejidad de la capacitación y del cumplimiento de la nómina y planificación.

ALUMNOS INSCRIPTOS. Es el conjunto de alumnos que se encuentran inscriptos en un curso determinado. No todos los alumnos inscriptos van a realizar la capacitación, sino que solo un subconjunto de éstos lo harán, ya que para cada capacitación existe un límite de cantidad de alumnos, determinado por las condiciones establecidas por la dirección, así como también por la capacidad áulica que va a contener a los alumnos.

AULA. Determina en qué lugar físico se va a realizar el dictado del curso. La clase Aula va a ser tratada más adelante en el texto.

ASISTENCIA. Es el conjunto de fechas y alumnos que determina quien asiste a cada una de las clases. Este conjunto de datos va a ser tratado más adelante en el texto. Los alumnos que tienen una asistencia regular a la capacitación, estarán en condiciones de rendir las prácticas y las evaluaciones. A su vez, para poder recibir la certificación de un curso es necesario cumplir con el 80% de asistencia.

EGRESADOS. Son un conjunto de alumnos que cumplen con todos los requisitos para aprobar el curso y recibir la certificación. Esta clase se va a tratar más adelante en el texto.

Clase PERSONA. El diseño que se implementó para el manejo de los datos de las personas consta de una generalización en la clase persona y tres subclases, alumnos, profesores y administrativos. Dentro de la clase Persona se encuentran algunos de los datos más característicos utilizados para realizar las estadísticas más importantes al momento de analizar la información interna de la institución. Primeramente, se detallará la clase Persona y luego las subclases.

NOMBRES. Nombres de una persona

APELLIDOS. Apellidos que posee una persona.

TIPO DE DOCUMENTO. Tipo de documentación que posee una persona. Los tipos más usuales son DNI, Pasaporte, Libreta de Enrolamiento y Precaria.

DOCUMENTO. Número de documentación declarado por una persona.

SEXO. Sexo de la persona.

CUIT. Número de CUIT de la persona. Este campo es exigido para la realización de algunos trámites y certificados con otros organismos.

TELEFONO CELULAR. Número de celular de contacto de una persona. En la actualidad es muy utilizado para la creación de grupos en redes sociales para la distribución de información y apuntes.

TELEFONO FIJO. Número de teléfono fijo de una persona. Utilizado para el caso de urgencias y accidentes en donde se debe ubicar un familiar o dejar un mensaje.

DATOS DE NACIMIENTO. Son los datos que involucran la fecha de nacimiento, el lugar y la nacionalidad de la persona, está declarada en una clase aparte y se detallará más adelante en el texto.

DIRECCION. Domicilio de una persona. Se encuentra en una clase aparte y se tratara más adelante en el texto.

DATOS DIGITALES. Datos de contacto electrónico de una persona. Se trata en una clase aparte y se detalla más adelante en el texto.

OCUPACION. Ocupación de una persona y condición laboral. Estos datos se tratan en una clase que se detalla luego en el texto.
PLAN SOCIAL. Indica si la persona posee un plan social y cuál es, para informar a organismos complementarios la regularidad de la persona y la asistencia al establecimiento.

Clase ALUMNO. La información que se maneja de los alumnos es la siguiente.

NUMERO DE ALUMNO. Número que identifica a un alumno en un determinado curso. El número de alumno para un alumno es diferente según la capacitación realizada.
ESTADO DE INSCRIPCION. Identifica el estado de un alumno en el listado de inscriptos. El estado puede ser Titular o Lista de Espera. En este último caso, un alumno puede ser llamado si un alumno titular no concurre al establecimiento.
ESTADO DE ALUMNO. Identifica el rendimiento del alumno a lo largo del curso. Si es un alumno regular, es un alumno que asiste esporádicamente o no se presenta a rendir las prácticas y evaluaciones.
MES DE BAJA. Es el mes en que el alumno decidió no continuar con la capacitación. Esto puede suceder por voluntad propia del alumno o por reiteradas inasistencias.
ALUMNO EN FICHA DE CURSO. Este atributo indica si el alumno se encuentra o no en el listado oficial del curso, dependiendo de las consideraciones del profesor a cargo del curso.
OBSERVACIONES DEL CURSO. Texto con una descripción sobre el desempeño de un alumno en un curso. Esta información es provista por el profesor del curso.

Clase PROFESOR. La información que se maneja de los profesores es la siguiente.

CUPOF. Es una clave que identifica al profesor dentro de los organismos provinciales.
ESTADO ACTUAL. Es el estado del instructor en la actualidad. Actualmente todos los profesores son declarados como provisionales hasta que se realicen los concursos por un cargo titular.

FECHA DE DESIGNACION. Fecha en la cual el profesor inicia las actividades en el establecimiento. Es la fecha indicada en la declaración jurada y en la toma de posesión al cargo docente.

ESPECIALIDAD. Especialidad en la cual el profesor se desempeña en la institución, determina el área en la cual tiene competencia.

POSEE FI. Indica si el profesor tiene el certificado de Formación de Instructores o lo tiene pendiente.

PUNTAJE. Detalla cual es el puntaje del profesor en el sistema. Este valor debe coincidir con el puntaje anual designado a cada uno de los profesores.

TITULO BASE. Es el titulo con el cual se va a desempeñar como profesor en el área correspondiente dentro del establecimiento y el que referencia su experiencia en el área mencionada.

CURSOS DICTADOS. Es el conjunto de cursos que dicta en el establecimiento. Es una colección de la clase Curso detallada anteriormente.

Clase ADMINISTRATIVO. La información que se maneja del personal administrativo es la siguiente.

CARGO. Cargo que ocupa esta persona en el establecimiento.

CUPOF. Es una clave que identifica al personal administrativo del establecimiento dentro de los organismos provinciales.

ESTADO ACTUAL. Estado del personal administrativo. Actualmente todos son declarados como provisionales hasta que se realicen los concursos por cargos titulares.

FECHA DE DESIGNACION. Fecha en la cual el personal administrativo inicia las actividades en el establecimiento. Es la fecha indicada en la declaración jurada y en la toma de posesión al cargo.

POSEE FI. Indica si el personal administrativo tiene el certificado de Formación de Instructores o lo tiene pendiente.

TITULO BASE. Título con el cual se va a desempeñar como administrativo en el establecimiento y el que referencia su experiencia en el área mencionada.

TURNO. Turno en el cual desempeña funciones dentro del establecimiento. Puede tomar tres valores: mañana, tarde o noche.

HORARIO. El horario determina los días y horarios en que cumple funciones el personal administrativo. La clase Horario se va a desarrollar más adelante en el texto.
ESPECIALIDAD. Declara la especialidad que tiene el personal administrativo para prevenir un cambio de funciones o cuando se realizan tareas solapadas y se necesitan varios administrativos en determinados turnos.

Clase ASISTENCIA. Como se hizo mención anteriormente existe una clase asistencia en donde se registra la asistencia de los alumnos para disponer de un nivel de presencialidad por cada aula y controlar tanto la deserción como la no finalización de los cursos. En especial se controla que la asistencia no baje del 80% y que la cantidad de alumnos asistentes no se encuentre por debajo de una cantidad determinada de alumnos regulares en cada capacitación. Es importante destacar que el control del presentismo favorece a mantener el nivel educativo que existe en cada curso. En caso de un escaso nivel de presencialidad, se procede a un análisis para establecer las causas de esta situación y generar los cambios pertinentes para mejorar los cursos. Los datos que se guardan son:

FECHA ACTUAL. Fecha en la que se pasa la asistencia.
ESTADO. Estado del alumno en la fecha controlada (presente o ausente).
ALUMNO. Alumno al cual se le está pasando asistencia.

Clase AULA. Las principales características de las aulas son detalladas a continuación.

NUMERO DE AULA. Identifica unívocamente a cada una de las aulas.
ESTADO DE AULA. Estado del mantenimiento del aula. Puede tomar el valor bueno, intermedio o malo.
OBSERVACIONES. Indica si el aula está dentro del establecimiento educativo o se encuentra fuera del mismo y en qué condiciones.
METROS CUADRADOS. Dimensiones del aula en metros cuadrados.
NOMBRE. Nombre del aula.
ESPECIALIDAD. Detalla para que especialidad (taller) está preparada el aula.

Clase DATOS DIGITALES. Los datos digitales sirven para contactar a los alumnos por diferentes medios e involucrarlos con las diferentes actividades que la institución ofrece. Además, sirven para compartir información a través de correo electrónico, redes sociales, videos, papers, presentaciones de diapositivas y demás herramientas que hacen a la comunicación con los alumnos un canal de aprendizaje en el cual muchas veces se encuentra fuera del horario de estudio. Se busca involucrar a los alumnos en diferentes momentos, no solamente cuando están presenciando clases, y de esta manera es más factible llegar a los alumnos en todo momento. Los datos digitales son los siguientes:

EMAIL DE CONTACTO. Email de contacto de un alumno, profesor o administrativo. Utilizando el email se logra compartir diferentes tipos de documentos y videos sobre los trabajos prácticos realizados.
PAGINA WEB. Indica la url de un sitio con el cual se puede interactuar entre las diferentes personas y compartir sus trabajos personales.
RED SOCIAL. Indica la red social preferente de un usuario (generalmente Facebook o Instagram) en donde promociona sus trabajos, practicas, o simplemente comparten información.
ESTADO DE DATOS DIGITALES. Indica la frecuencia de intercambio de información por medios digitales. Es decir, si los alumnos o personal los utilizan frecuentemente o terminan siendo una vía de comunicación sin interacción por parte del destinatario.

Clase DATOS DE NACIMIENTO. Involucra los datos asociados al nacimiento de una persona. El rango de edad de los alumnos generalmente indica los tipos de curso que suelen tomar. En la mayoría de los cursos orientados a oficios de rama pesada, como pueden ser herrería, soldadura, refrigeración, entre otros, lo rangos de edad suelen ser superiores a los 30 años. Para dicho rango de edad, por ejemplo, es más dificultosa la comunicación vía redes sociales. Resulta más conveniente comunicarse mediante videos y o practicas presenciales extracurriculares. La clase Datos de Nacimiento dispone de los siguientes datos:

DIA DE NACIMIENTO. Día de nacimiento de una persona.

MES DE NACIMIENTO. Mes de nacimiento de una persona.
AÑO DE NACIMIENTO. Año de nacimiento de una persona.
LUGAR DE NACIMIENTO. Localidad de la Provincia de Buenos Aires, Provincia, u “Otro” si la nacionalidad no es Argentina.
NACIONALIDAD. Nacionalidad de una persona.

Clase DIRECCION. Esta clase tiene un domicilio actualizado de las personas que componen el instituto de formación con el fin de determinar cuáles son las zonas de más impacto sobre las que se está trabajando y cuáles son las zonas donde se debería llegar con más presencia para generar más emprendimientos productivos. El análisis de estos datos permite llegar a lugares inesperados con una relativamente baja capacidad de difusión, y por eso es una actividad que se realiza año tras año con el simple objetivo de tener cada vez mayor alcance a la población. La clase Dirección dispone de la siguiente información:

CALLE PRINCIPAL. Calle en donde se ubica el domicilio de la persona
CALES SECUNDARIAS. Calles que intersecan a la calle principal de manera perpendicular y a ambos lados.
NUMERO. Número que dispone el domicilio para determinar su ubicación exacta.
DEPARTAMENTO. Número de departamento, para el caso en que la persona viva en un complejo habitacional.
PISO. Numero de piso, para el caso en que la persona viva en un complejo habitacional vertical.
LOCALIDAD. Localidad en donde se encuentra ubicado el domicilio de la persona.
PROVINCIA. Provincia en donde se encuentra ubicado el domicilio de la persona.
CODIGO POSTAL. Código postal de la localidad de la persona.

Clase EGRESADO. Se dispone de un listado de personas que egresaron de los diferentes cursos para posteriores consultas, así como también para la realización de las certificaciones correspondientes y las constancias de la finalización.

ALUMNO. Persona que egresa un curso

NOTA FINAL. Promedio de notas obtenidas a lo largo de las prácticas realizadas, así como también de las evaluaciones confeccionadas por el alumno.
NUMERO DE EGRESADO. El número de egresado indica cual es el número que tiene el alumno respecto al curso aprobado. Es un número que determina la aprobación y va a ser transcripto al libro de egresados en el cual los alumnos se notificaran fehacientemente para la entrega de la certificación correspondiente.
CURSO. Indica el curso que aprobó el alumno.

Clase HORARIO. Almacena información sobre los días y horarios en que se dictan los cursos, así como también en qué momento asisten las diferentes autoridades administrativas. Esta clase es usada también para determinar las aulas que están ocupadas en un momento determinado, crear un organigrama áulico y así poder organizar los mantenimientos correspondientes. Permite llevar un control de las personas que desempeñan tareas administrativas y se puede verificar en qué momento están realizando dichas tareas y en que turno.

LUNES. Valor binario que indica presencia o no.
MARTES. Valor binario que indica presencia o no
MIERCOLES. Valor binario que indica presencia o no.
JUEVES. Valor binario que indica presencia o no.
VIERNES. Valor binario que indica presencia o no.
HORARIO LUNES. Indica el horario del día correspondiente.
HORARIO MARTES: Indica el horario del día correspondiente.
HORARIO MIERCOLES: Indica el horario del día correspondiente.
HORARIO JUEVES: Indica el horario del día correspondiente.
HORARIO VIERNES: Indica el horario del día correspondiente.

Clase OCUPACION. Mediante esta clase es posible hacer un seguimiento de las diferentes tareas laborales desempeñadas por las personas. A partir de este análisis se intenta verificar si el desempeño laboral de los alumnos ha cambiado en el tiempo gracias al conocimiento adquirido en el instituto. Es una forma de verificar el impacto de las capacitaciones y a su vez determinar si los alumnos aplican el conocimiento

adquirido en el ámbito laboral, en relación de dependencia o de forma autónoma, o simplemente realizan las capacitaciones por interés propio.

OCUPACION PRIMARIA: Detalla la ocupación principal de una persona. Es uno de los campos que más se hace énfasis a lo largo del año tanto de manera individual como de manera colectiva. (Declarada como String la variable).
OCUPACION SECUNDARIA. Dato que complementa a la ocupación primaria. Permite analizar casos como los de las personas que desean dedicarse a su actividad secundaria y luego de realizar una capacitación esta actividad se transforma en su ocupación primaria.
OCUPACION TERCIARIA. Complementa los anteriores y solamente sirve como una referencia, y para determinar la cantidad de ocupaciones que puede tener una persona.
REGIMEN IMPOSITIVO. Indica cual es el régimen principal en que la persona se encuentra contenida y sirve como dato estadístico. Los regímenes pueden ser en relación de dependencia, autónomo, monotributista, entre otros.

Clase USUARIOS. Involucra los datos de usuario y contraseña.

USUARIO: Indica el usuario del sistema.
CONTRASEÑA. Indica la contraseña de dicho usuario.

Clase INSTITUCION EDUCATIVA. Esta clase contiene la información referente a la institución. Son datos que identifican a la institución dentro de los organismos correspondientes, así como también datos para dar conocimiento a la población de la existencia de la institución. La mayoría de los datos son utilizados en diferentes trámites administrativos, desde los más sencillos hasta algunos que llevan meses de aprobación. Este era uno de los problemas principales del manejo de la información en formato papel, ya que muchas veces se cargaban los datos de forma incorrecta y se dificultaba la aprobación de este tipo de trámites. Por ejemplo, para un alumno egresado es muy importante obtener la certificación oficial de una capacitación finalizada, y muchas veces debían esperar varios meses debido a errores presentes en formularios en papel.

NOMBRE. Indica el nombre de la institución educativa y es de conocimiento público. Con este nombre las personas pueden identificar a la institución en redes sociales y distintos medios de comunicación.

JERARQUIA. Detalla la jerarquía que tiene la institución para el organismo oficial. Este valor depende del lugar en donde se encuentra la institución, como así también de la cantidad de horas cátedra que posee y de las personas que trabajan en la misma.

NUMERO. Indica el número de institución. Este número es determinado por el organismo oficial en donde se encuentra registrada la institución a nivel provincial.

INSTITUCION CONVENIANTE. Indica la institución u organismo por el cual fue creado el establecimiento y que lo respalda tanto a nivel estructural como sociocultural. Se trata de un organismo centralizado que funciona de respaldo. en el caso que los organismos oficiales no puedan crear establecimientos educativos de nivel formativo laboral.

REGICE. Es un numero que identifica el establecimiento a nivel provincial.

CUE. Es la clave única de establecimiento educativo e identifica al mismo para la realización de trámites administrativos a nivel provincial y nacional.

TELEFONO DE CONTACTO. Número telefónico oficial para la comunicación tanto con los alumnos como con los organismos oficiales.

ESPECIALIDAD. Indica la especialidad para la cual está declarado el establecimiento y engloba la principal oferta formativa.

HORARIO. Indica los días y horarios para los cuales el instituto de formación está abierto para la realización de las capacitaciones.

DIRECCION. Domicilio del establecimiento.

DATOS DIGITALES. Datos de contacto digitales, redes sociales y emails de comunicación.

AULAS. Conjunto de aulas que componen el establecimiento.

PROFESORES. Conjunto de profesores que componen el plantel docente del establecimiento.

EGRESADOS. Conjunto de egresados que dispone el establecimiento a lo largo del tiempo.

ADMINISTRATIVOS. Conjunto de personal administrativo que desempeña funciones en el establecimiento educativo.

4.3 – TECNOLOGIA UTILIZADA PARA EL DESARROLLO DE LA APLICACIÓN

Al momento de definir la tecnología adecuada para el proyecto se analizaron las alternativas disponibles con el fin de seleccionar una opción que sea robusta y no quede obsoleta con el paso del tiempo. De esta forma se podría acompañar el crecimiento de la entidad educativa y las exigencias que pudieran surgir sobre la manipulación de los datos almacenados.

Para el desarrollo de la aplicación se eligió el lenguaje de programación Java, por ser un lenguaje utilizado para múltiples propósitos y su adaptabilidad a las necesidades de un backend robusto. Además, es un lenguaje muy conocido y utilizado en toda la industria del software, cuenta con una gran comunidad y soporte, y es escalable.

Una vez determinado el uso del lenguaje de desarrollo Java se optó por Hibernate como la alternativa adecuada de mapeador objeto relacional, ya que es uno de los ORM más utilizados y el de mayor conocimiento por parte del desarrollador y autor de este trabajo. Hibernate cuenta con una extensa documentación y una gran cantidad de tutoriales sobre configuración, utilización y puesta en funcionamiento.

Como motor de bases de datos se utilizó MySQL, por su sencillez, versatilidad, fácil configuración y licencia libre, además de ser un producto muy utilizado. Asimismo, resulta muy estable y simple de mantener para bases de datos medianas que no manejan grandes volúmenes de datos.

4.4 – DETALLE DE LOS MAPEOS REALIZADOS

Los mapeos de las distintas clases fueron realizados en archivos XML. Cada archivo contiene la información necesaria para mapear los datos de una clase en la base de datos final. A continuación, se muestra el contenido de los archivos XML de cada clase.

Mapeo de la clase CURSO. A continuación, se muestra el mapeo implementado para clase Curso.

```
<class table="curso" name="Curso">
   -<id name="oid" column="idCurso" type="long"><generator class="increment"/></id>
   <property name="nombreDelCurso" column="nombreDelCurso" type="string" length="100"/>
```

```
   <property name="certificacion" column="certificacion" type="string" length="100"/>
   <property name="anioCicloLectivo" column="anioCicloLectivo" type="long"/>
   <property name="tipoDeCapacitacion" column="tipoDeCapacitacion" type="string" length="100"/>
   <property name="numeroCurso" column="numeroCurso" type="long"/>
   <property name="fechaInicio" column="fechaInicio" type="string" length="100"/>
   <property name="fechaFinalizacion" column="fechaFinalizacion" type="string" length="100"/>
   <many-to-one name="profesor" column="Profesor" unique="true" not-null="true" cascade="all" />
   <many-to-one name="horario" column="Horario" unique="true" not-null="true" cascade="all" />
   <property name="cantidadDeHoras" column="cantidadDeHoras" type="long"/>
   <property name="modalidad" column="modalidad" type="string" length="100"/>
   -<set name="alumnosInscritos" cascade="all"><key column="idCurso"/><one-to-many
class="Alumno"/></set>
   <many-to-one name="aula" column="Aula" unique="true" not-null="true" cascade="all" />
   -<set name="asistencia" cascade="all"><key column="idCurso"/><one-to-many
class="Asistencia"/></set>
   -<set name="egresados" cascade="all"><key column="idCurso"/><one-to-many
class="Egresado"/></set>
</class>
```

Mapeo de la clase ALUMNO. Para el mapeo de la clase Alumno se tuvo en cuenta la conjunción de los datos persona y los del alumno para que compongan todos en una misma tabla como se muestra a continuación.

```
-<class table="administrativo" name="Administrativo">
   -<id name="oid" column="idAdministrativo" type="long"><generator class="increment"/></id>
   <property name="nombres" column="nombres" type="string" length="100"/>
   <property name="apellidos" column="apellidos" type="string" length="100"/>
   <property name="tipoDni" column="tipoDni" type="string" length="100"/>
   <property name="dni" column="dni" type="long"/>
   <property name="sexo" column="sexo" type="string" length="100"/>
   <property name="cuit" column="cuit" type="string" length="100"/>
   <property name="telefonoCelular" column="telefonoCelular" type="string" length="100"/>
   <property name="telefonoFijo" column="telefonoFijo" type="string" length="100"/>
   <many-to-one name="datosDeNacimiento" column="DatosNacimiento" unique="true" not-
null="true" cascade="all" />
   <many-to-one name="direccion" column="Direccion" unique="true" not-null="true" cascade="all" />
   <many-to-one name="datosDigitales" column="DatosDigitales" unique="true" not-null="true"
cascade="all" />
   <many-to-one name="ocupacion" column="Ocupacion" unique="true" not-null="true" cascade="all"
/>
   <property name="planSocial" column="planSocial" type="string" length="100"/>
   <property name="fichaMedica" column="fichaMedica" type="string" length="100"/>
   <property name="tratamientoMedico" column="tratamientoMedico" type="string" length="100"/>
   <property name="educacionAlcanzada" column="educacionAlcanzada" type="string" length="100"/>
   <property name="observaciones" column="observaciones" type="string" length="100"/>
   <property name="numeroAlumno" column="numeroAlumno" type="long"/>
   <property name="estadoDeInscripcion" column="estadoDeInscripcion" type="string" length="100"/>
   <property name="estadoDeAlumno" column="estadoDeAlumno" type="string" length="100"/>
   <property name="mesBajaAlumno" column="mesBajaAlumno" type="string" length="100"/>
   <property name="alumnoEnFichaDeCurso" column="alumnoEnFichaDeCurso" type="string"
length="100"/>
   <property name="observacionesDelCurso" column="observacionesDelCurso" type="string"
length="100"/>
```

```
</class>
```

Mapeo de la clase PROFESOR. Para el mapeo de la clase Profesor se tuvo en cuenta la conjunción de los datos persona y los del profesor para que compongan todos en una misma tabla como se muestra a continuación.

```
-<class table="profesor" name="Profesor">
   -<id name="oid" column="idProfesor" type="long"><generator class="increment"/></id>
   <property name="nombres" column="nombres" type="string" length="100"/>
   <property name="apellidos" column="apellidos" type="string" length="100"/>
   <property name="tipoDni" column="tipoDni" type="string" length="100"/>
   <property name="dni" column="dni" type="long"/>
   <property name="sexo" column="sexo" type="string" length="100"/>
   <property name="cuit" column="cuit" type="string" length="100"/>
   <property name="telefonoCelular" column="telefonoCelular" type="string" length="100"/>
   <property name="telefonoFijo" column="telefonoFijo" type="string" length="100"/>
   <property name="observaciones" column="observaciones" type="string" length="100"/>
   <property name="cupof" column="cupof" type="string" length="100"/>
   <property name="estadoActual" column="estadoActual" type="string" length="100"/>
   <property name="fechaDesignacion" column="fechaDesignacion" type="string" length="100"/>
   <property name="poseeFI" column="poseeFI" type="string" length="100"/>
   <property name="tituloBase" column="tituloBase" type="string" length="100"/>
   -<set name="cursos" cascade="all"><key column="idProfesor"/><one-to-many class="Curso"/></set>
   <many-to-one name="ocupacion" column="Ocupacion" unique="true" not-null="true" cascade="all"
/>
   <many-to-one name="datosDeNacimiento" column="DatosNacimiento" unique="true" not-
null="true" cascade="all" />
   <many-to-one name="direccion" column="Direccion" unique="true" not-null="true" cascade="all" />
   <many-to-one name="datosDigitales" column="DatosDigitales" unique="true" not-null="true"
cascade="all" />
</class>
```

Mapeo de la clase ADMINISTRATIVO. Para el mapeo de la clase Administrativo se tuvo en cuenta la conjunción de los datos persona y los del personal administrativo para que compongan todos en una misma tabla como se muestra a continuación.

```
-<class table="administrativo" name="Administrativo">
  -<id name="oid" column="idAdministrativo" type="long"><generator class="increment"/></id>
   <property name="nombres" column="nombres" type="string" length="100"/>
   <property name="apellidos" column="apellidos" type="string" length="100"/>
   <property name="tipoDni" column="tipoDni" type="string" length="100"/>
   <property name="dni" column="dni" type="long"/>
   <property name="sexo" column="sexo" type="string" length="100"/>
   <property name="cuit" column="cuit" type="string" length="100"/>
   <property name="telefonoCelular" column="telefonoCelular" type="string" length="100"/>
   <property name="telefonoFijo" column="telefonoFijo" type="string" length="100"/>
   <property name="planSocial" column="planSocial" type="string" length="100"/>
   <property name="fichaMedica" column="fichaMedica" type="string" length="100"/>
```

```
   <property name="tratamientoMedico" column="tratamientoMedico" type="string" length="100"/>
   <property name="educacionAlcanzada" column="educacionAlcanzada" type="string" length="100"/>
   <property name="observaciones" column="observaciones" type="string" length="100"/>
   <property name="cargo" column="cargo" type="string" length="100"/>
   <property name="cupof" column="cupof" type="string" length="100"/>
   <property name="estadoActual" column="estadoActual" type="string" length="100"/>
   <property name="fechaDesignacion" column="fechaDesignacion" type="string" length="100"/>
   <property name="poseeFI" column="poseeFI" type="string" length="100"/>
   <property name="tituloBase" column="tituloBase" type="string" length="100"/>
   <property name="turno" column="turno" type="string" length="100"/>
   <many-to-one name="ocupacion" column="Ocupacion" unique="true" not-null="true" cascade="all"
/>
   <many-to-one name="datosDeNacimiento" column="DatosNacimiento" unique="true" not-
null="true" cascade="all" />
   <many-to-one name="horario" column="Horario" unique="true" not-null="true" cascade="all" />
   <many-to-one name="direccion" column="Direccion" unique="true" not-null="true" cascade="all" />
   <many-to-one name="datosDigitales" column="DatosDigitales" unique="true" not-null="true"
cascade="all" />
</class>
```

Mapeo de la clase ASISTENCIA. A continuación, se muestra el mapeo implementado para clase Asistencia.

```
-<class table="asistencia" name="Asistencia">
   -<id name="oid" column="idAsistencia" type="long"><generator class="increment"/></id>
   <property name="fechaActual" column="fechaActual" type="string" length="100"/>
   <property name="estado" column="estado" type="boolean"/>
   <many-to-one name="alumno" column="Alumno" unique="true" not-null="true" cascade="all" />
</class>
```

Mapeo de la clase AULA. A continuación, se muestra el mapeo implementado para clase Aula.

```
-<class table="aula" name="Aula">
   -<id name="oid" column="idAula" type="long"><generator class="increment"/></id>
   <property name="numeroAula" column="numeroAula" type="long"/>
   <property name="estadoAula" column="estadoAula" type="string" length="100"/>
   <property name="observaciones" column="observaciones" type="string" length="100"/>
   <property name="metrosCuadrados" column="metrosCuadrados" type="long"/>
   <property name="nombre" column="nombre" type="string" length="100"/>
   <property name="especialidad" column="especialidad" type="string" length="100"/>
</class>
```

Mapeo de la clase DATOS DIGITALES. A continuación, se muestra el mapeo implementado para clase Datos Digitales.

```
-<class table="datosDigitales" name="DatosDigitales">
```

```
   -<id name="oid" column="idDatosDigitales" type="long"><generator class="increment"/></id>
   <property name="emailContacto" column="emailContacto" type="string" length="100"/>
   <property name="paginaWeb" column="paginaWeb" type="string" length="100"/>
   <property name="redSocial" column="redSocial" type="string" length="100"/>
   <property name="estadoDeDatosdigitales" column="estadoDeDatosdigitales" type="string"
length="100"/>
</class>
```

Mapeo de la clase DATOS DE NACIMIENTO: A continuación, se muestra el mapeo implementado para clase Datos de Nacimiento.

```
-<class table="datosNacimiento" name="DatosNacimiento">
   -<id name="oid" column="idDatosNacimiento" type="long"><generator class="increment"/></id>
   <property name="diaNacimiento" column="diaNacimiento" type="long"/>
   <property name="mesNacimiento" column="mesNacimiento" type="long"/>
   <property name="anioNacimiento" column="anioNacimiento" type="long"/>
   <property name="lugarDeNacimiento" column="lugarDeNacimiento" type="string" length="100"/>
   <property name="nacionalidad" column="nacionalidad" type="string" length="100"/>
</class>
```

Mapeo de la clase DIRECCION. A continuación, se muestra el mapeo implementado para clase Direccion.

```
-<class table="direccion" name="Direccion">
   -<id name="oid" column="idDireccion" type="long"><generator class="increment"/></id>
   <property name="callePrincipal" column="direccionCallePrincipal" type="string" length="100"/>
   <property name="callesSecundarias" column="direccioCallesSecundarias" type="string"
length="100"/>
   <property name="numero" column="direccionNumero" type="long"/>
   <property name="localidad" column="localidad" type="string" length="100"/>
   <property name="provincia" column="provincia" type="string" length="100"/>
   <property name="codigoPostal" column="codigoPostal" type="long"/>
</class>
```

Mapeo de la clase EGRESADO. A continuación, se muestra el mapeo implementado para clase Egresado.

```
-<class table="egresado" name="Egresado">
   -<id name="oid" column="idEgresado" type="long"><generator class="increment"/></id>
   <many-to-one name="alumno" column="Alumno" unique="true" not-null="true" cascade="all" />
   <property name="nota" column="Nota" type="long"/>
   <property name="numeroEgresado" column="numeroEgresado" type="long"/>
   <many-to-one name="curso" column="Curso" unique="true" not-null="true" cascade="all" />
</class>
```

Mapeo de la clase HORARIO. A continuación, se muestra el mapeo implementado para la clase Horario.

```
-<class table="horario" name="Horario">
```

```
  -<id name="oid" column="idHorario" type="long"><generator class="increment"/></id>
  <property name="lunes" column="lunes" type="boolean"/>
  <property name="martes" column="martes" type="boolean"/>
  <property name="miercoles" column="miercoles" type="boolean"/>
  <property name="jueves" column="jueves" type="boolean"/>
  <property name="viernes" column="viernes" type="boolean"/>
  <property name="horarioLunes" column="horarioLunes" type="string" length="100"/>
  <property name="horarioMartes" column="horarioMartes" type="string" length="100"/>
  <property name="horarioMiercoles" column="horarioMiercoles" type="string" length="100"/>
  <property name="horarioJueves" column="horarioJueves" type="string" length="100"/>
  <property name="horarioViernes" column="horarioViernes" type="string" length="100"/>
</class>
```

Mapeo de la clase OCUPACION. A continuación, se muestra el mapeo implementado para clase Ocupación.

```
-<class table="ocupacion" name="Ocupacion">
  -<id name="oid" column="idOcupacion" type="long"><generator class="increment"/></id>
  <property name="ocupacionPrimaria" column="ocupacionPrimaria" type="string" length="100"/>
  <property name="ocupacionSecundaria" column="ocupacionSecundaria" type="string"
length="100"/>
  <property name="ocupacionTerciaria" column="ocupacionTerciaria" type="string" length="100"/>
  <property name="regimenImpositivo" column="regimenImpositivo" type="string" length="100"/>
</class>
```

Mapeo de la clase USUARIOS. A continuación, se muestra el mapeo implementado para clase Usuarios.

```
-<class table="usuario" name="Usuario">
  -<id name="oid" column="idUsuario" type="long"><generator class="increment"/></id>
  <property name="usuarioDelSistema" column="usuarioDelSistema" type="string" length="100"/>
  <property name="contrasenia" column="contrasenia" type="string" length="100"/>
</class>
```

Mapeo de la clase INSTITUCION EDUCATIVA. A continuación, se muestra el mapeo implementado para clase Institución Educativa.

```
-<class table="institucionEducativa" name="institucionEducativa">
  -<id name="oid" column="idInstitucionEducativa" type="long"><generator class="increment"/></id>
  <property name="nombre" column="nombre" type="string" length="100"/>
  <property name="jerarquia" column="jerarquia" type="string" length="100"/>
  <property name="numero" column="numero" type="long"/>
  <property name="institucionConveniante" column="institucionConveniante" type="string"
length="100"/>
  <property name="regice" column="regice" type="string" length="100"/>
  <property name="cue" column="cue" type="string" length="100"/>
  <property name="telefonoDeContacto" column="telefonoDeContacto" type="string" length="100"/>
  <property name="especialidad" column="especialidad" type="string" length="100"/>
  <many-to-one name="horario" column="Horario" unique="true" not-null="true" cascade="all" />
  <many-to-one name="direccion" column="Direccion" unique="true" not-null="true" cascade="all" />
```

```
   <many-to-one name="datosDigitales" column="DatosDigitales" unique="true" not-null="true"
cascade="all" />
   -<set name="aulas" cascade="all"><key column="idInstitucionEducativa"/><one-to-many
class="Aula"/></set>
   -<set name="cursos" cascade="all"><key column="idInstitucionEducativa"/><one-to-many
class="Curso"/></set>
   -<set name="profesores" cascade="all"><key column="idInstitucionEducativa"/><one-to-many
class="Profesor"/></set>
   -<set name="egresados" cascade="all"><key column="idInstitucionEducativa"/><one-to-many
class="Egresado"/></set>
   -<set name="administrativos" cascade="all"><key column="idInstitucionEducativa"/><one-to-many
class="Administrativo"/></set>
</class>
```

4.5 – ESQUEMA DE LA BASE DE DATOS

En la figura 8 es posible observar el equema de tablas de la base de datos resultante del proceso de mapeo.

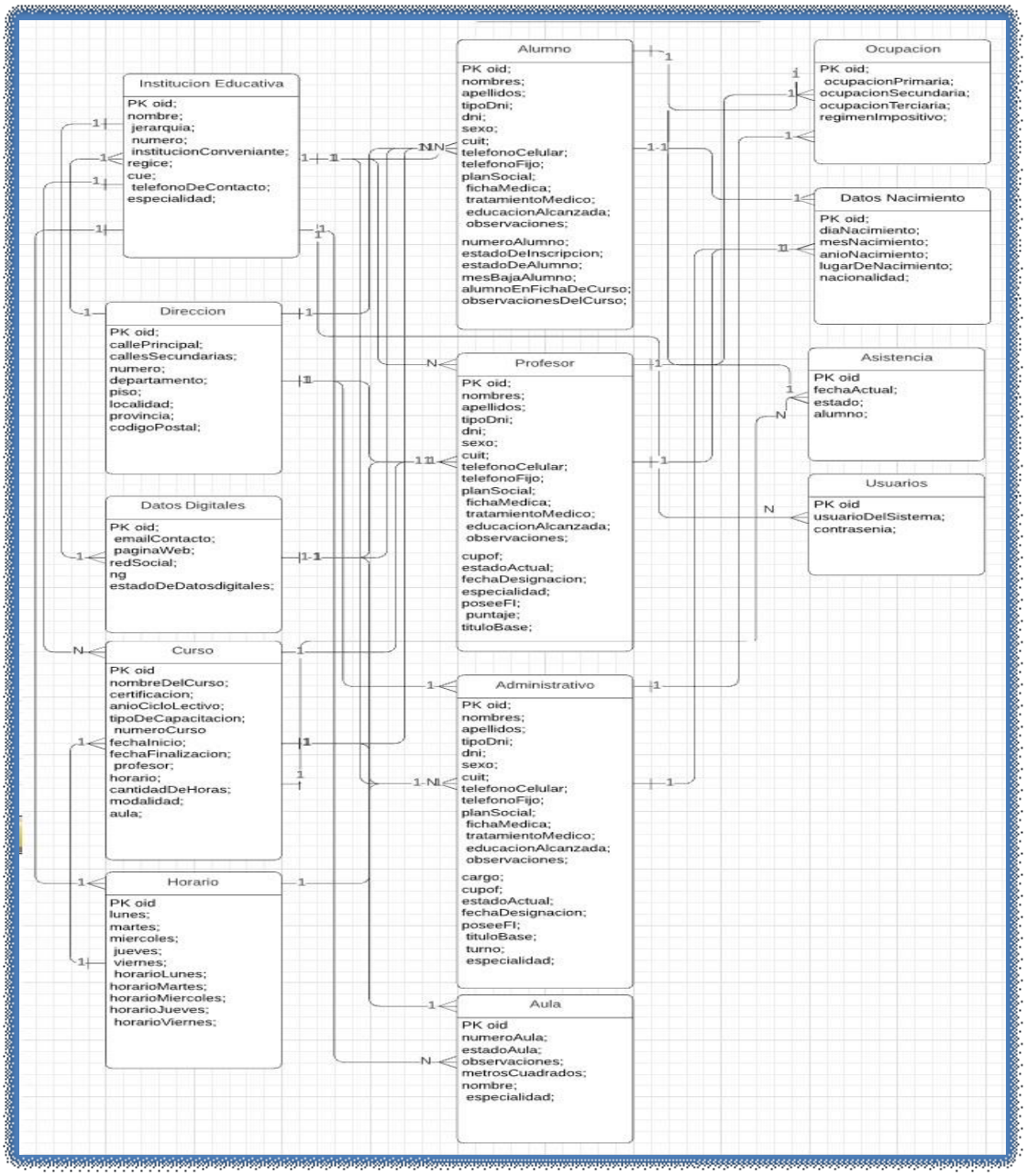

Figura 8. Equema de tablas de la base de datos.

5 – CONCLUSIONES Y OBJETIVOS A FUTURO

Se realizó una investigación sobre las alternativas de ORM existentes en el mercado. Además, se realizó un análisis comparativo de tres herramientas de mapeo objeto-relacional existentes, determinando las ventajas y desventajas de cada una. Luego, se determinó la herramienta de mapeo objeto-relacional adecuada para el caso de estudio.
Se desarrolló una aplicación de gestión de información que permitió acompañar el crecimiento de un instituto que brinda distintas capacitaciones para el fortalecimiento de los oficios y el establecimiento de mecanismos de autoempleo para aquellas personas que no encuentran lugar en el mercado laboral. La aplicación presenta una mejora tanto en el manejo de la información administrativa como en la información que asiste al proceso de toma de decisiones.
A su vez, el manejo de los datos de contacto digitales promueve y facilita el acercamiento de los alumnos a la institución. Mediante el análisis de datos estadísticos se determina a qué sectores de la población el instituto aún no está abarcando, y se aumenta la convocatoria en dichos sectores, fortaleciendo los canales de publicidad y así generando un interés formal de potenciales alumnos, sin esperar a que tomen la decisión de acercarse al establecimiento educativo. Además, la posibilidad de permanecer en contacto con el conjunto de alumnos mediante redes sociales, compartiendo documentos y videos resulta de gran utilidad e incrementa la continuidad de los alumnos.
La aplicación desarrollada permite además mejorar la planificación de las capacitaciones, integrándolas entre sí, y logrando que los alumnos cuenten con los conocimientos necesarios para encontrar ocupación en el mercado laboral una vez que egresan.
Los objetivos a futuro en el proyecto se centran en dos puntos importantes. El primero de ellos es el de actualizar el sistema con el uso de nuevas técnicas de ORM. Esto involucraría, por ejemplo, la inclusión y uso del framework Spring. Por otro lado, es necesario agilizar la gestión del presentismo de alumnos, profesores y personal administrativo. Una posibilidad de realizar esto es mediante lectores de huellas dactilares o algoritmos de reconocimiento facial, reduciendo aún más el uso de medios

en formato papel. De esta forma se mantendría la información de presentismo actualizada en todo momento y se liberaría a los profesores de realizar este tipo de tareas, promoviendo un mejor aprovechamiento del tiempo en clase.

BIBLIOGRAFIA

1. DB-Engines. DB-Engines Ranking. https://db-engines.com/en/ranking
2. DB-Engines. Ranking of Relational DBMS. https://db-engines.com/en/ranking/relational+dbms
3. DB-Engines. Ranking of Object Oriented DBMS. https://db-engines.com/en/ranking/object+oriented+dbms
4. Jeffrey M. Barnes. Object-Relational Mapping as a Persistence Mechanism for Object-Oriented Applications. 2007.
5. Hibernate. https://hibernate.org/
6. MyBatis. https://mybatis.org
7. TopLink. https://www.oracle.com/middleware/technologies/top-link.html
8. ObJect Relational Bridge (OJB). https://db.apache.org/ojb
9. Cayenne. https://cayenne.apache.org/why-cayenne.html
10. PBeans. http://pbeans.sourceforge.net/
11. SimpleORM. http://www.simpleorm.org/
12. Kodo JDO. https://docs.oracle.com
13. jOOQ. https://www.jooq.org
14. ActiveJDBC. https://javalite.io/activejdbc
15. Spring Data. https://spring.io/projects/spring-data