Review on JPA Based ORM Data Persistence Framework

Neha Dhingra, Emad Abdelmoghith, and Hussien T. Mouftah

Abstract— **Large scale application design and development involve some critical decisions. One of the most important issue that affects software application design and development is the technology stack used to develop such large systems. System response time measures how quickly an interactive system responds to user input. Programming tools like Object Relational Mapping (ORM) is used to handle the communication between object model and data model components which is vital for such systems. Currently, Hibernate is considered the most flexible ORM frameworks and has become the de facto standard for JPA-based data persistent frameworks. This article reviews the most widely used ORM providers, especially frameworks that provide support for Java Persistence API (JPA) like Hibernate JPA, EclipseLink, OpenJPA and Data Nucleus.**

Index Terms—**Hibernate, eclipselink, openjpa, data nucleus.**

I. Introduction

Object/Relational Mapping (ORM) is a technique to transmute data from an object-oriented model into the relational database model. Object-Oriented Programming (OOPs) is based on entities, whereas the relational databases management system (RDBMS) sordid on relations and fields to store data. Fig. 1 shows the mapping process between the java classes and the relations database. Interpretation of the java entities into the relational database requires interoperability among the disparate architectures. In order to obviate mapping worriment, ORM bridges the gap between the plat- forms and manage the disparity between the object graphs and the structured query language (SQL). For a developer, segregated mapping layer deprecates the complexity of the boilerplate code [1]. ORM wraps the functionality of an old conventional Java Database Connectivity (JDBC) programming model [2] into the persisted databases. A conventional ORM application propounds a lightweight object-oriented interface called the Data Access Object (DAO) [2]. A DAO layer determines the designing pattern that encapsulates the java entities into a sequence of SQL operations (e.g. Insert, Delete or Update) through predefined functions. To execute a query and retrieve the relational data efficiently in the object-oriented programming, a language called DQL (Doctrine Query Language) [3] was introduced to reduce the complexity of the user by simple data definition language (DDL) commands. DQL is a distinguishable platform to retrieve the java entities using predefined set of protocols. Apart from the obvious programming convention, ORM accelerate the optimization process through transaction locking and maintain data writes through defined transactional [3] boundaries. Moreover, ORM attunes data accessed in a record-based patterns. ORM standardized the persistence process as through the java persistence API (JPA) interface. JPA is a java application programming interface [4] that manages the data between the java objects and the relational databases. JPA is a specification, not an implementation to persist data in the RDBMS. Due to the failure of the enterprise persistence model and lack of java persistence standard, developers often materialized JPA implementations as an attempt to optimize the mapping architecture. JPA implementations increase the portability and extensibility of the code, by de-coupling the JPA specifications from the underlying API architecture. The next couple of sections discuss a comparison based on JPA and JPA implementation, which would decompose the view of a developer to formalize the approach while developing an API [1]-[4].

Manuscript received March 6, 2017; revised June 16, 2017.

Neha Dhingra, Emad Abdelmoghith, and Hussien T. Mouftah are with the Department of Electrical Engineering and Computer Science, University of Ottawa, Canada (e-mail:{ndhin017, eabdelmo and mouftah}@uottawa.ca).

II. JPA Providers

Java Persistence API (JPA) is an interface that persists the java entity to the relational database [5]. A JPA specification is a set of empty methods and collection of interfaces that only describe java persistence methodologies and provides standardized programming through the JPA implementation. According to Ogheneovo *et al*. JPA is a standard-compliant framework defined for mapping plain old java object (POJO) into the relational databases. Currently, most of the JPA persistence providers have released several commercial [5] and open [5] source JPA implementations. For instance Hibernate by JBOSS and RedHat [6], EclipseLink by Oracle and sun glassfish project [7], OpenJPA by IBM and Bea [8] and Data Nucleus by JPOX and Tapestry [9]; are some of the commercially available and vendor independent providers that rigorously follow JPA paradigm in order to configure an API.

Developing an API based on the appropriate JPA implementation is determined by three potentials prospects. Firstly, it is the compatibility between the relational database and the JPA provider, which is based on the complexity of the SQL operations i.e triggers, indexes, stored procedures. Second, it is the precipitancy of building a prototype which is based on the affordance of the API that means the ease at which a developer of an API performed operations. Finally, the middlewares [9] adopted, while building the mapping strategy. For example, JBoss is a middleware software for Hibernate API. According to Miki Enoki *et al.* [9] middleware is software that combines the software component or enterprise application; it is a layer that lies between the OS and the API. To map the data into database,

DOI: 10.7763/IJCTE.2017.V9.1160

JPA performs metadata modeling and schema creation which then is accomplished by standard annotations by defining @annotationname or via the XML files using tags. Annotations are acknowledged as an exemplary alternative to XML because of the ease of programming. Both mapping techniques follow similar functionality but due to the high complexity in the XML, it is not preferred much in the programming of an API. For example, an annotation/XML tag to create a primary key column in an entity is defined by @ID in the annotations and in XML the syntax is <name="ID" value="integer"/>. A typical, JPA API defines run-time interface objects to persist data and to create a connection. JPA exemplify boilerplate code at run-time by instantiation of pre-defined interfaces. To create a connection between the java objects and the database, JPA defines an interface object called the EntityManagerFactory [10]. This object performs allocation and de-allocation of the resources. Once the mapping process is completed, the manager is destroyed and the resources are detached. An EntityManagerFactory object is invoked by the object of an EntityManager interface. The EntityManager then interact within a persistence context to perform create, read, update and delete (CRUD) operations on the entities.

The persistence process in JPA manages the SQL operation through the transaction and query objects to retrieve and execute data definition language (DDL) operations. Fig. 2 outlined a three layers high-level architecture where the top-most layer is the graphical user interface (GUI) that communicates with the client and server in order to perform operations on the front-end. The middle layer is the JPA Layer; it is divided into the service layer using controllers, data access objects (DAO), repositories and service implementation, the second layer in the JPA layer is called the data persistence layer. The persistence layer defines the mapping procedure between the relational database and the java entities. The bottom layer determines the type of RDBMS in order to persist data in the tabular format. According to [10] a typical JPA implementation also support spatial and geographical data storage for persistence [5]-[10].

A. Hibernate

Hibernate is a vendor independent ORM framework; it maps the java object-oriented model into the relational database by directly persisting access between the plain old java objects and the relational database. According to [11]; hibernate manages to provides high performance, feature rich mapping technology to persist java data types into the underlying structured query language (SQL) data types. For example, an integer field in java class is converted into int (integer) column in the Microsoft SQL Server. Hibernate being a comprehensive solution accomplish persistence by not only taking responsibility for mapping the java entities to the database tables but also overcoming the development time contrarily wasted on binding a java API through java database connectivity (JDBC). In a study by Bhushan S. Sapre *et al*. in discussed that Hibernate's mapping process utilize run-time persistence properties to create an association between the java classes and the relational fields, in order to avoid impedance mismatch problem.

Because hibernate is a free software it is distributed under the GNU Lesser General Public License 2.1 [12]; it is an optimal solution for resource-oriented APIs. The mapping process in Hibernate is based on the lazy load, which means all the necessary information about the generating the schema, creating a sub for a java source file creating primary-foreign key relationships between the entities. In the same study Bhushan S. Sapre *et al.* in [12] also stated that hibernate follows high- level abstraction by encapsulating the underlying architecture of the API from the developer, which diminishes the complexity of the code to fewer methods and interfaces. The process increases the extensibility of the code but constrains the knowledge of the developer to abstract methodologies. Another study by Nisha Sharma *et al.* in [13] on Hibernate manages to provide a details assessment on queries to the database through Hibernate Query Language (HQL) [13]. HQL is an advanced query language to retrieve objects from the java classes and execute complex SQL operation.

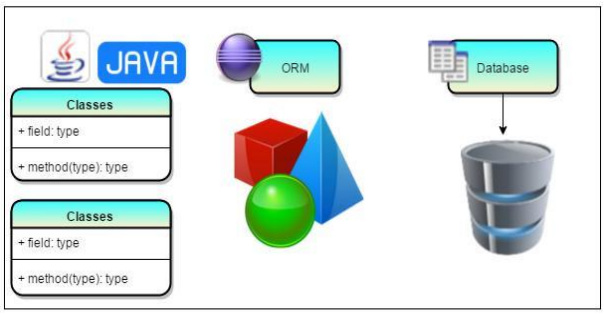


Fig. 1. Java ORM architecture.

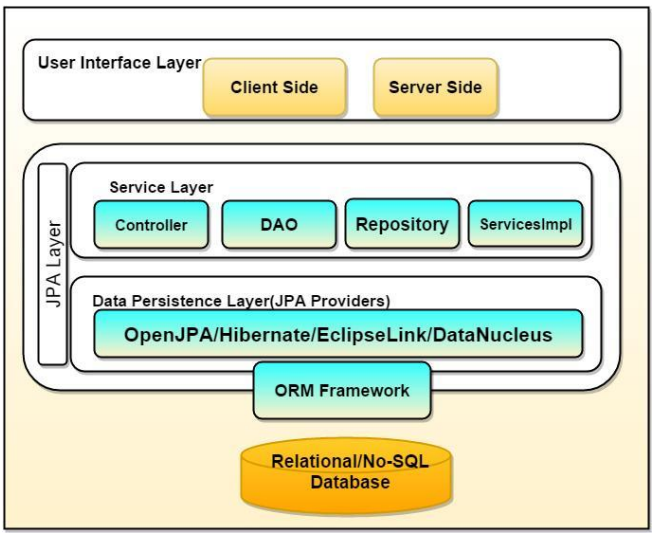


Fig. 2. JPA architecture.

In another study by B.vasavi *et al*. [14]; stated that configuration object in Hibernate is the first object, which comprises of the properties about the connection to be configured. The configuration object addresses and authenticates the properties of Hibernate by creating a connection to the database. A configuration file is used by the SessionFactoryManager to start the persistence process and create the connection between the java entities and the database tables. The ses- sionFactoryManager object is invoked then by the session instance to execute SQL operation. In this process the session, object maintains the cache and needs an explicit close() method in order to avoid

memory leaks. To perform transactions on the database, a session object is used to invoke the transaction object. The transaction objects in Hibernate execute operations in a particular session through transaction manager and the underlying query. In another study by B.Vasavi *et al.* [14] also stated that Hibernate allows developers to create an efficient business application either using annotations or XML files. In annotations, the functionalities are injected to map the java entities into the relational tables whereas in XML the same operation is performed through complex tags. Because of the flexibility in practicing annotations, Hibernate manages to override the existing mapping scenarios by mapping enum into the columns of the database and also outline a single property in the java entity to multiple columns in the back- end (RDBMS). In order to standardized Hibernate API, JPA created standard interfaces to implement programming that is platform independent. Hibernate API is a fully complaisant with Technology Compatibility Kit (TCK) [15] to accomplish database modeling in JPA. According to [14] with the rapid advancement of the web application, Hibernate has been receiving a great consideration as a platform independent portable language along with the spring framework; it pro- vides a persistence layer for a complex enterprise application with spring that act as a template to encapsulate Hibernate sessions into the template methods [15]. According to Jorge Edison Lascano, Hibernate comprehend Persistence.XML configuration file to initialize the properties of a JPA project, using the org.hibernate.ejb.Hibernate Persistence provider. A typical Java Platform, Standard Edition (Java SE) through [16] JPA persist databases using the Entity Manager Factory object; it is a static method to initialize the persistence process. An Entity Manager Factory object is similar to the Session Factory Object in Hibernate. In an API, requests are made to invoke the java entities for persistence because the EntityManager performs the basic CRUD operations by invoking the object of Entity Factory Manager instance. The following Fig. 3 and Fig. 4 describes Hibernates high-level architecture to describe ORM mapping using the session or Entity Manager Object. After which Hibernate configures connection management using the Entity Manager Factory or Session Manager Factory object. However, the transaction and query management objects in Hibernate provides the capabilities to perform run-time data retrieval operations to execute queries, but being an over ambitious process this creates memory overheads. According to the several studies and books on JPA 2.1, Hibernate JPA facilitates additional functionality to a native API through advanced object-oriented programming. These OOPS concepts enhanced the functionality of a conventional API by handling inheritance and polymorphism through the implicit super class mapping and platform independent Java Persistence Query Language (JPQL) [11]-[16].

B. EclipseLink

In a study by Lukas Sembera [2] on JPA implementation, it was reviewed that EclipseLink's advanced database extensions through the SQL compliant features such as stored procedures, native SQL queries, and indexes improved the efficiency of the API. EclipseLink or Eclipse Persistence Service project [14] is a vendor independent performance oriented ORM solution. EclipseLink being a sophisticated JPA provider started as Oracle's TopLink product and was adopted by java community as an API. The evolution of EclipseLink as an open source solution has been enabling the developers to build an efficient application. In [15] Doug Clarke asserted that currently EclipseLink provides ORM mapping solution with JPA, by binding Object-XML Moxy (with support for JAXB) and SDO (Service Data Objects) methods and interfaces. These mapping process increases the reliability of the API by allowing the developer to learn one language and perform tasks in every implementation based on the same platform. Therefore, EclipseLink JPA performs persistence process through extended annotations or XML files, using org.eclipse.persistence.annotations [15] packages. In a similar study in [15] it was stated that EclipseLink also delivers not one but a set of runtime API services through the EntityManager class, which persist in the Javax.persistence [16] package. The persistence process in EclipseLink is based on built-in libraries in Eclipse IDE, which decreases the complexity to manage JAR files explicitly and also diminishes the compatibility issues in the API. EclipseLink being defined as a comprehensive JPA ORM solution which as stated in [16] delivers persistence services to the developers through an efficient applications that access the data in a variety of formats and data sources. Because EclipseLink has been an experienced advanced ORM mapping technique EclipseLink JPA manages the complex relational databases operations progressively. In a similar study Doug Clarke also added that mapping techniques have evolved over the period of 12 years in EclipseLink resulting in a domain model to create a detailed schema sordid on cloud. EclipseLink being a commercial as well as open-source framework sup- port advanced tools which are independent of the under- lying implementation. Furthermore, EclipseLink also sup- ports extended runtime API through command-line with packages like org.eclipses. persistence.pa [16]. Apart from existing features, EclipseLink JPA also supports advanced functionalities such as coordinated shared cache, clustered databases on RDBMS or Non-relational database. Weaving not only added an enhancement process in the API to enhance the java entities in order to retrieve data from and into the database, but it also configures the enhancements of the java classes to track for lazy loads and changes in the fetch data groups to perform internal optimization [16]. By Default, EclipseLink supports run-time enhancement because it allows the entities to efficiently enforce primary key-foreign key [16] relationships and manage joins among two or more entities. In another study on EclipseLink in [17]; developers managed coordinated shared caching in EclipseLink, which allowed consistency among distributed APIs with multiple persistence units. The mapping process in EclipseLink according to [17] is divided into following classes; A Project class: It is a container to persists mapping and configures metadata. In addition, to existing JPA interface support, EclipseLink weaving technique simplified the manipulation process by using bytecode [18] which were processed at run-time or compile time. A Descriptor: It contains mapping information for each data member that EclipseLink should

persist to or transform to. A Map file: It includes a project.XML file, and is associated with sessions that EclipseLink can use at run-time [17] for bytecode enhancement. Being a standard JPA implementation EclipseLink uses the EntityManagerFactory Object to create connection and the EntityManager object to map the java entities to the relational database for CRUD operation. Fig. 5 describes an EclipseLink JPA high-level architecture.

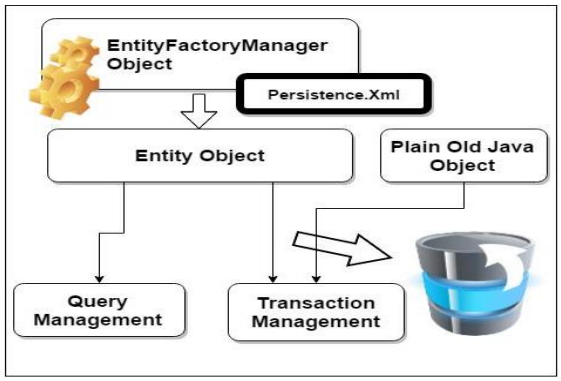


Fig. 3. Hibernate JPA ORM architecture.

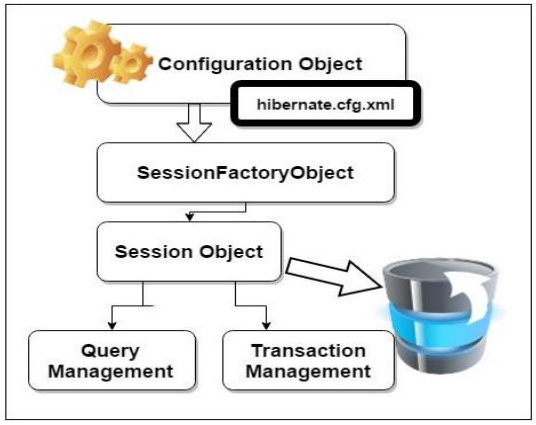


Fig. 4. Hibernate ORM architecture.

The persistence in EclipseLink configures the connection with the EntityManager object which is then invoked by the EntityManagerFactory object. Configuring a connection in EclipseLink is a straightforward process wherein the objects of Transaction and Query interfaces are used to retrieve and execute complex SQL operations based on the EntityManager object. EclipseLink also includes additional capability called weaving while persisting data into the database. This technique manages schema creation and map the java classes into the relational databases to increase the efficiency of the API [2], [15]-[18].

C. OpenJPA

In a study by Lukas Sembera in [2] it was stated that OpenJPA allows developers to generate user de- fined sequences by implementing seq interface. These Sequence generator interfaces are used as primary key columns in an entity. OpenJPA is an open source light-weighted JPA implementation that is integrated with Apache server [17]. While caching in OpenJPA lack generality it has two level; data cache that retrieves the entities loaded from the databases and query cache that stores the primary-key column returned in the transaction for reference [17]. Developing an OpenJPA API automates the mapping procedure and the schema creation using SynchronizeMapping [17] in the persistence.XML file. The generator interfaces also in-clude additional capabilities to store time in the primary key through the TimeseededSeq [18] and random hex string using the UUIDHexSeq [18].

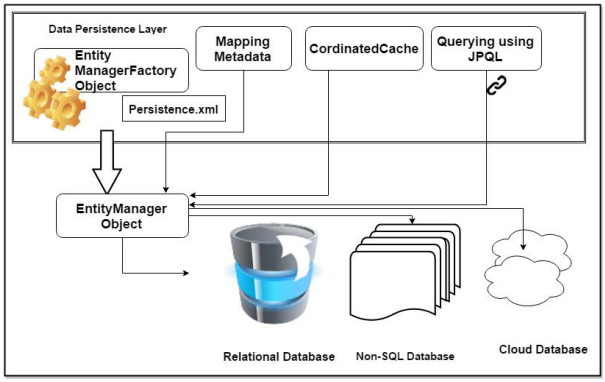


Fig. 5. EclipseLink JPA architecture.

In [19] it was asserted that by default, OpenJPA domain model is incompetent in handling the entity constraints. However, OpenJPA's explicitly reconstruct constraints using the SchemaFactory [19] property. According to OpenJPA's designers, a standard JPA imple- mentation essentially depends on the monitoring of java entities, but the specification does not define how to im plement these monitoring [19]. However, some OpenJPA providers auto-generate innovative sub-classes or proxy objects [20] to improve the monitoring process explicitly, which are created on the entity objects at the run-time. While other JPA providers utilize byte-code weaving technologies to enhance the actual entity class objects automatically. OpenJPA manages run-time enhancements, by creating sub-classes, which works subtle with the small sized API but degrades performance and create functionality defects in a real-time production environment. Another problem in OpenJPA was stated in a study by Miki Enoki *et al*., in which it was proposed that caching in OpenJPA is coarse- grained level which results in a low cache hit rate [19]. An improved caching strategy was introduced in [19] that adjusted indexes with the fixed size at the granular level. The process improved the performance of the cache by referencing objects dynamically [19]. But due to some defects, in the creating dynamic and automatic index where the frequency of updates were high this approach failed. Thus, researchers are still working in the OpenJPA's cache management to create fixed size index in the persistence layer. The process has dividing the cache into levels and increased the cache coordination between the java entities. However, when the frequency of search queries are high to the database caching manages CRUD operations before flushing the content of the database. The second level caching supplements an extra layer in OpenJPA by getting an overview of a particular cache object and then eliminating the need to explicitly handle developer's request.

According to [18] an OpenJPA implementation persist java classes either using annotations with property declarations including @column, @id and @version [19] or using XML files. The @column indicates the name of the

column, @id defines the unique identifier similar to primary key and @ver- sion annotation is common practice to ensures the data integrity during merges and acts as an optimistic concur- rency control [20]. Apart from the API, a developer has to configure jar files such as Java. OpenJPA.All [20] and Javax.persistence [20] packages. The persistence process also instantiates the objects of EntityFactoryManager and EntityManager to complete the persistence process. Entity-FactoryManager Object is a JPA standard implementation, to create a one-time connection while the EntityManager wraps dynamic transactions for persistence. The diagram in Fig. 6 reviews a high-level architecture of OpenJPA by defining the persistence process using the JPA standard specifications through the EntityManger and the EntityFactoryManager Object. The diagram also incorporates two-level cache strategies in OpenJPA mainly the Query Cache and the Data Cache to optimize the API efficiency [2], [17]-[20].

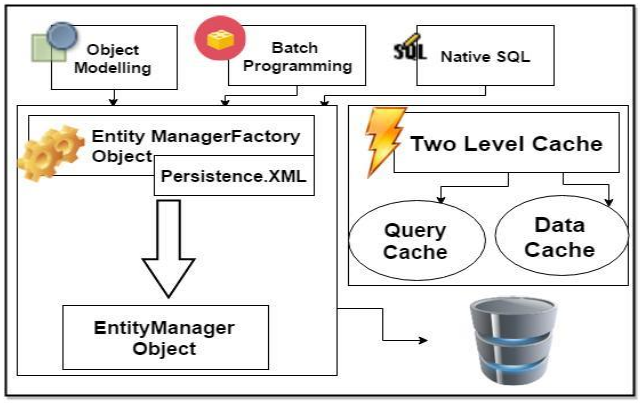


Fig. 6. OpenJPA architecture.

D. Data Nucleus

Data Nucleus is an open source java persistent framework that the saves state [21] of the java Object into the relational database. But entity graphs in data nucleus cache data through generic compilation [21]; they are an initial step in the Data Nucleus caching strategy because they are independent of the underlying data source and create expression tree [21]. The next process to cache the java entities creates a ”compiled expression tree” [21], which is then are converted into the native language of the relational database. Finally, the executed query returns the object cached with the ”resultant” objects According to the Data Nucleus designing community, Data Nucleus is a fully complaisant JPA implementation [21] providing transparent data persistence and byte- code enhancement to persist data in the database. Through transparent data persistence, Data Nucleus directly manipulate data stored in the relational database using the java entities. In Data Nucleus, the persistence process is divided into two stages. In the first phase enhancement process utilizes the common technique of bytecode manipulation to create persisted java class [21]. Data Nucleus manages the enhancement using” Data Nucleus Enhancer”; it performs manipulation on the java classes by creating metadata [21]. In the second phase the persistence process performs a schema formulation through the “schema tool”; which generates a metadata file and use annotations to persist the java entities. Data Nucleus Schema tool accomplishes persistence through the “data nucleus.autocreateSchema” property that is defined in the persistence.XML file. The schema genera- tion process in Data Nucleus is dependent on the”enhancer” to create metadata for every class and sub-class in the API. In Data Nucleus, a metadata file contains information about all the persistable units of the API. Thus, increasing the efficiency of the mapping process and allowing the developer to perform abstract process by simple method calling. This programming style is useful in distributed environment although it is complicated in the centralized architecture.

A major distinction between the other JPA implementations and Data Nucleus is the built-in support for schema creation. Data Nucleus provides flexibility to entreated schema manually, through the command prompt or through java programming using annotations or XML files. The entire pro-cess is highly sensitive along with the addition of duplicate jar files. The compatibility issues in Data nucleus could restrain the creation of metadata, in order to avoid problems in metadata creation and managing schema a developer usually configure the enhancements manually. According to the data nucleus designer, a data nucleus API must includes following jar files to complete the persistence process i.e. datanucleus-core-2.1.1.jar, data nucleus-enhancer- 2.1.0-release.jar, data nucleus-JPA 2.1.0-release.jar*, datanu-cleusrdbms2.1.1.jar and asm-3.1.jar * [21] respectively. In a study by Miroslav Nachev [21] on JPA implementations; it was stated that Data Nucleus manages persistence process in an incremental fashion, which mean not all fields are retrieved immediately. The process is evolved into step as to improve the efficiency of the overall application through the lazy loading process called the entity graph. These graphs are further managed by standard JPA interfaces called the EntityManagerFactory objects and the EntityManger. All the SQL operation such as create, delete update and select are managed by the EntityManager.

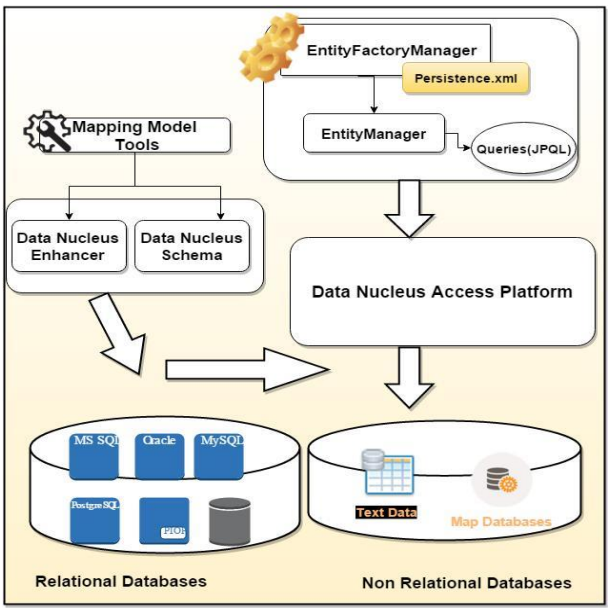


Fig. 7. DataNucleus JPA architecture.

Fig. 7 shows a high-level DataNucleus architecture with EntityManager objects to perform persistence. A typical JPA

implementations persist classes defined via metadata [21]. This metadata is defined in the form of traditional XML files or annotations. The XML files are a highly preferred mode of object map- ping because it involves invincibly less deployment time to retrieve and store data into the database. Although data nucleus doesn't define a mechanism to cache a query [21].

III. Comparison of JPA Providers

A. Connection and Configuration

Configuring a Connection in a JPA is accomplished by setting few properties in the persistence.XML file and using the objects of the EntityManager and the Hibernate with its built- in and customized connection pool setup through advanced protocols

TABLE I: Connection and Configuration Features

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
Connection Pool	Build-in + Third party Open source (c3p0, proxool)	Not Default ,Build-in (max/min 32 with initial 1 connection)	Need Third party+ Plug-in support.	Not Default but third party data source DBCP, C3P0, Proxool, Bone CP, JNDI, and lookup connection Data Source.
Configure connection Appropriately	Need to set up hibernate. cfg.xml file with properties.	Configure connection Property Add-on Connection Retry option.	Default: on-demand	Three auto start options at classes, xml, schema table .
Detached State Manager	Supports Transient, Persistent and Detached Objects in Hibernate.	Explicit Detach, cascade Detach ,Bulk Detach	Off By Default, pros needs enhanced persistent classes and the OpenJPA libraries at client tier.	Data Nucleus fetch groups to control the specific fields to detach.
Always Close Resources	Session need explicit closing, Bean close automatically.	Need to close explicitly or will create Memory Leak Problem.	Garbage collection+ application level cleaning.	Auto managed by Data Nucleus

Such as C3P0 [8] and protocol [8] provides the most optimized connection. Alternatively, EclipseLink, Open-JPA, and Data Nucleus are dependent on a third party tool. Configuring the state of an API is a complex task. JPA implementation such as Data nucleus's advance functionality fetches groups of data to control fields at the granular level. EntityManagerFactory interfaces. According to Jorge Edison Lascano [10], a java persistence API exfoliates adequately in any environment, no matter whether it is an in-house intranet that serves a few number of users or for developing a demanding API that attends thousands of users. JPA manages the connection implicitly through the EntityManagerFactory which handles the bottleneck to open, close a connection and detached or attach the allocated resources. In the similar study, Jorge Edison Lascano [10] also asserted that JDBC executes a cache pool in order for the application to avoid opening connections to the database respectively. To save time a connection pool executes the query in data cache to implement persistence in the JPA providers. This means that resources such as memory, cache, and hard disk fetches the data in the persistence process as a single persistence unit at runtime. Every JPA implementation is assigned a set of connection protocols and fetching strategies to optimize the resource utilization process. Table I compares connectivity and configuration protocols and default values in all four JPA-implementations; it deter- mines, which JPA implementation provides the most flexibility by customizing connection or by setting properties and also include information about the default values that are designated on a particular API. Other implementation i.e. Hibernate and EclipseLink inefficiently complicate the task by creating complex decoupling in de-allocation process [8], [10].

B. Cache

Caching process in JPA enhances the performance of the API during inflow and outflow of data or to execute SQL operations on the database. According to Jorge Edison Lascano in [10], it is stated that caching in JPA is implemented through a series of constant updates, and the developers do not have to bother about the refreshing the cache or any comparable activities related to caching or flushing. JPA manages cache through distributed cache frameworks [10] i.e. ecache, memcached, cacheman [10] or other JPA implementations; therefore a developer does not have to allocate time in completing the cache services. According to [10], caching process is divided into levels in JPA; the first level or L1 cache is described in the persistence context (for a session or EntityManger) while the second level or L2 cache is called the coordinated or shared cache [22]. To improve the performance of the cache, strategies defined in [22] to create adjustable indexes that would improve caching technique in JPA but the problem nevertheless exist where the frequency of updated were high in the API. And the third level is implemented in Data Nucleus only in the form of a result tree [23] to generate the query in the persistence layer. In a study by Miki Enoki *et al.* caching in JPA implementation are, managed through a flushing technique that re-initialize the internal SQL cache and executes the command creating cache invalidation problem. In Hibernate query cache is responsible for caching the output of SQL queries through the primary keys [23]. In

EclipseLink entity cache [23] require a third party assistance to cache data. Similarly, for openJPA and data nucleus query optimizer explicitly manages the cache when the query executes. Table II below on Caching in JPA differentiate between all four JPA specification by comparing various caching features [10], [22], [23].

TABLE II: CACHING FEATURES

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
Flushing	Auto by default, Other mode Commit, manual, never, Always	Default (off) flush clear.cache+Drop, DropInvalide, Merge	Automatically flush before queries involving dirty objects will ensure that this never happens.	flush.mode to AUTO (default) but allow manual handling of "n" objects.
Data Cache(2 level)	clustered cache, JVMlevel (SessionFactorylevel) cache on a class-by-class and collection- by-collection, using any strategy : read-write cache, non strict-read-write, transactional cache	Cache with no locking, no cache refresh. -session cache(default)+ query + cache (size , .invalidation) By default, + Eclipse Link caches objects read from a data source	Data and Query caching (optional cache). -Not related to the Entity Manager cache. - Data cache can operate in both single- JVM and multi-JVM environments.	By default the Level 2 Cache is enabled + mode of operation of the L2 cache default UNSPECIFIED , others include ENABLE _ SELECTIVE , DISABLE _SELECTIVE, ALL, NONE.
Utilize the EntityManager cache	Default 1st level cache Add-on 2nd level and Query cache.	Default is not shared (Entity manager)+ shared object cache option in EclipseLink.	RetainState configuration option to true, using build in cache.	Need to explicitly close connection detachOnClose to set to True.
Query Cache	Default Disabled, To Turn set Property to values=True	No Option Need to be configured	Default Disabled, To Turn set Property to values=True +supports Concurrent Query Cache.	Generic Compilation Include a tree which is database independent.

C. Query and Transaction

According to Jorge Edison Lascano in [10] every Transaction either retrieve or send data to the database independent of the underlying data source. In JPA to query entities through the java object, we use the Java Persistence Query Language (JPQL). JPQL is a case sensitive language queries which ex- ecute the SQL operations using the java objects. In [10] Jorge Edison Lascano also stated, that JPA manages large data-set optimally by reducing the SQL code and thus avoid the SQL injections by executing the code at runtime. Every JPA implementation follows a build-in default fetching strategy or customized databases extraction strategies thus eliminating the need to manage and build fetching models. A Transaction signifies as a unit of work performed within the relational database management system (RDMBS) [24], following the basic relational reliable policies called ACID property [24] (atomicity, consistency, isolation, and durability). A JPA specification identifies every transaction as an integral part of mapping process and executes the queries on the entities to retrieve data from the database. In order to understand the

Complex features in JPA implementation and compare transactional factors to help developers understand the functionality and perform SQL operations Table III provides complete detailed analysis. The comparison indicates that OpenJPA provides a wide range of options for querying databases, but due to high bug issues in the language, other JPA implementation such as hibernate and EclipseLink are more preferred in terms of optimized result [10], [21], [24].

TABLE III: QUERY AND TRANSACTION FEATURES

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
Transactions Optimization	Fetch optimization techniques and patterns. with checkpoints.	Change Tracking for Transactions.	Aggregates and projections.	JPQL, NativeSQL +JDO Query.
Use Fetching	Lazy by default, can be set to eager. EAGER: Convenient, but slow. LAZY: More coding, but much more efficient.	Lazy Default but can make it eager.	Eager default fetch can be changed to Lazy. Strategy could be None, join , parallel.	JDO provides fetch groups, whereas JPA2.1 now provides EntityGraphs (A subset of fetch groups).
Query Parameters for encoding search data in filter Strings	Named parameters Need help!!	PERSISTENCE_UNIT_DEFAULT (which is true by default)	OpenJPA Aggressive caching of query compilation data, and the effectiveness of this cache is diminished if multiple query filters are used where a single is used.	All dirty objects are Flushed.
Large Data set Handling	Hibernate Pagination Hibernate ScrollableResultsNative SQL Each has its own advantage and disadvantage.	Pagination is one technique used in handling data sets.	By default, OpenJPA uses standard forward-only JDBC result sets, and completely instantiates the results of database queries on execution.	Native SQL, JPQL, JDOQL Allowing extensions for Query handling in large data set.
Query and Transaction Management	Manual Transaction Management. Can be automated with transaction Manager. PROPAGATION_ REQUIRED or Use HQL.	Work in unit of work from a session with isolation level. To Query use executeQuery •Nested Unit of Work •Parallel Unit of Work	JPQL +Extensions	Work in unit of work With Local transactions, JTA transactions, container managed transactions, spring managed transactions.
Tune fetch groups	Uses lazy select fetching for collections and lazy proxy fetching for single-valued.	Pre-defined fetch groups at the Entity+ Dynamic (use case) fetch groups at the query level Load all data and leave large fields (binary, additional join)	Load all data and leave large fields(binary, additional join)	Fetching objects with manual control to fetch.
Database indexes	Support Indexing	@Index annotation	Manual +Build-in	IndexMetaData+ optimization

D. Auto-Insert

Marking a field with the @GeneratedValue annotation confines the value of the field in the relational database to auto increment [25]. In a JPA implementation defining a primary key to uniquely identify a row in a relation uses auto, identity, sequence and table values. Every value specifies a behavioral pattern, wherein an auto adds special global number generator [25] in the ID column for every java entity. And an incrementor called the identity which auto generate values with an exception. The whole process is automated so its optimal and efficient to add a primary key column into the database. Table IV on JPA auto insert man- ages to compares all four implementations based one major factor called sequence increment; it depicts a comparison of the JPA implementations based on that increment factor for auto generation in the ID column [25].

TABLE IV: AUTO-INSERT FEATURES

JPAProviders Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
Sequence Increment	SequenceStyleGenerator With increment more then 1. With following options IDENTITY SEQUENCE (best option not much restriction) TABLE (SEQUENCE).	Sequence number pre-allocation enables a batch of ids to be queried from the database simultaneously in order to avoid accessing the database for an id on every insert. Default Value: 50.	Large bulk inserts Sequence overhead. own sequence factory can further optimize sequence number retrieval.	+Need to Set validate with Cache property to false. +auto identity generator is recommended. + sequence default can be non- optimum set key _ cache _ size= 10.

E. JPA Object

Java Persistence API has a collection utility packages with a wide range of embedded interfaces to create a list, set, collection, maps, tree maps [12] and many other calculative operations. A list and set are the most widely used utilities in order to perform the basic data retrieval task from the object model into the database model. Every JPA implementation supports certain default option while retrieval of the information from the database. Hibernate, Eclipselink, and Data Nucleus accomplish data retrieval through Set classes. Alternatively, OpenJPA operates on collections with an over-head in performance while retrieving the data. While sets are not considered an optimized solution in java method calling because of equals and hashCode methods in entities do not have the immutable functional key [12]. In [12] Doug Clarke stated that a list, without an index, in hibernate and eclipseLink is handled as a bag which degrades the performances of the API when the load increases. Table V shows a comparative difference among all 4 JPA implementations based on the list and set [25] and also differentiate which API utilize a set to execute the SQL operations [12], [25] .

TABLE V: JPA OBJECT CLASS FEATURES

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
use set instead of List / collections	Default used SET recommended by Hibernate creators. But option include List, Array, Map, bag and ibag.	Use set default but can add JPA class.	Default collection cause overhead, use Set, SortedSet, HashSet, or TreeSet.	Set is default for collection of data . can Use any other List, Set, Array, Map.

F. Threading

A multi-threaded standalone application persists data into the database and manages the threading issues to perform CRUD operations (create, read, update and delete). Table VI shows a comparison among difference JPA specification to manage threads in the multi-database environment. Every JPA implementation has an Entity Manager object to execute Query and an Entity Factory Manager to handle boilerplate code. However, Entity Manager is not threaded safe [26], which means we cannot create an object of the Entity Manager and perform transactions from the same instance. Entity Factory Manager, on the other hand, is synchronized which means that one object for EntityFactoryManager is managed throughout the API for de-allocation and allocation of the resources. Table VI show a comparison of all four JPA implementation. However, it also includes a comparison among JPA implementation based on distributed transaction (XA) in different APIs. Many blogs, tutorial, and prior research have discussed threading in detail and way to improve the performance of an API but JPA manages to achieve the accepted performance level [26].

TABLE VI: THREADING FEATURES

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
Multi-threading	Do not use hibernate managed objects in multiple threads. Settle for ID column	Handle but time consuming.	Single-thread default can be set using the openjpa.Multithreaded	Persistence Manager multithreaded. Default value is false.
XAtransaction XA(distributed transaction)	Hibernate Transaction Manager (searching)	Time out problem.	XA slower than standard transaction, but support non-xa and XA transaction.	Nothing available.

G. Mapping and Distributed Transactions

An Entity is an essential part of an API. In JPA objectmodeling is performed on objects called entities. These entities have relationships [27] defined among them. Mapping is an association between two or more entity where each onehas a role defined to create a relation. In an entity cardinality of the relation defines the constraint specially to the number of relationship. The JPA model maps a typical java class to the relational database sordid on the subclass and super-class relationship. In this paper, Table VII shows a comparison of all 4 JPA implementation with several optimization features and techniques such as; garbage collection, pagination, batch processing, auditing and logging to track relationships and manage the cardinality/ordinarily of the entity to improve the mapping process. Every implementation intuitively follows a rational approach. But because the relation between the entities has a bolder effect compared to the other factors it is important to tune the entities before the intricacy occurs. However, Hibernate support for OOPs concepts includes advanced features to accomplish mapping efficiently relation between the relation between the entities has a bolder-effect [28] compared to the other factors it is important to tune the entities before the intricacy occurs. However, Hibernate support for OOPs concepts includes advanced features to accomplish mapping efficiently. Whereas Open JPA and Data nucleus inherit single table operation, compared to join among table or a table per class strategies [27], [28].

TABLE VII: MAPPING AND DISTRIBUTED TRANSACTIONS FEATURES

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
Inheritance	Hibernate supports the three basic inheritance mapping strategies; table per class hierarchy, table per subclass, table per concrete class, concrete class strategy, concrete class using implicit polymorphism ConcretePolymorphism doesn't support join fetch.	Type of inheritance: Single Table Inheritance, Joined Table, Table per Concrete.	Mapping inheritance hierarchies to a single database table is faster for most operations than other Strategies employing multiple tables. Strategy SINGLE _TABLE, JOINED, or TABLE _ PER_CL ASS.	Need to must specify the identity of objects in the root persistable class of the inheritance hierarchy. You cannot redefine it down the inheritance tree. Default: single _ table . Other options: Joined, Table per class.
Composite Persistence	Hibernate+Jboss+sprig+ JPA will work.	Composite Persistence unit for relational and non relational database+ clustering.	No working 2013	Allow Run time persistence unit using JDO or JPA use the same persistence unit . Can't Find for Composite.

TABLE VIII: PERFORMANCE OPTIMIZATION FEATURES

JPA Providers Features	Hibernate	EclipseLink	OpenJPA	DataNucleus
JVM optimization	Garbage Collection	Java performance test suite.	Hotspot compilation modes and the maximum memory.	linked hashmap save 4% CPU time. And Hotspot is another option.
Preload Meta Data Repository	Doesn't have the option (search how it performs the same operation in hibernate).	MetadataSourceAdapter.	By default, the MetaDataRepository is lazily loaded which means fair amounts of locking. This option load metadata upfront and remove locking.	DataNucleus use JPA with Maven in pom.xml for Repository.
Enhancer	Bytecode enhancer(run and compile time) Maven, Ant , Gradle.	Weaving (run +compile time)	build-time or deploytime enhancement. post - compilation bytecode enhancer.	Default enhancement before runtime. Support Transparent Persistence -Run +Compile Time.
Enable logging/Disable	Enable for performance analysis Log4jjdbc + jbosslogging (warn, error and fatal).	(eclipselink.logging.level) Values (Off, severe, warning , info, config , fine, finer, finest, all) This is an optimization feature that lets you tune the way EclipseLink detects changes in an Entity. Default Value: AttributeLevel if using weaving (Java EE default), otherwise Deferred.	verbose logging affects performance.	Log4J +set categories : Persistence Transaction Connection Query Cache, Metadata, Data- Source, schema native Schema-tool, JPA IDE Value Generation Recommended: DataNucleus category to OFF.
Logging Performance Tracking/Auditing	Default Envers for tracking old version, individual Entity properties (New).	ChangeTrackingType; ATTRIBUTE, OBJECT or DEFERRED+ Auditing Ways. AUDIT_USER and AUDIT_TIMESTAMP column. Full history support.	JDBC performance tracker(set to false).	MBeans internally to track changes via JMX at runtime Or It own API for Monitoring.

H. Performance Optimization

Performance Optimization in JPA implementation contributes some of the major criterion's to develop an API; it includes an optimizer, fetch strategies, indexes and parameterized searching options [28]. To bridge different implementations, JPA includes pre-loaded metadata repositories. Performance Optimization in JPA implementation contributes some of the major criterion's to develop an API; it includes an optimizer, fetch strategies, indexes and parameterized searching options. To bridge different implementations, JPA includes pre-loaded metadata repositories. These repositories are useful in order to perform tasks, such as metadata mapping and improving query response time. JPA implementations support pre-compiled mapping through MetedataAdpters [28]. These adapters are pre-loaded in the API to automate the metadata creation [28]. EclipseLink JPA includes a MetadataSourceAdapter [29] to implement the mapping process, whereas other implementations such as OpenJPA and Hibernate are inefficient in loading the repositories. These repositories are useful in order to perform tasks, such as metadata mapping and improving query response time. JPA implementations support pre-compiled mapping through MetedataAdpters [30]. Table VIII compares JPA implementations comparing features affecting performance [28]-[30].

IV. Conclusion

JPA implementations acknowledge the programmers to build extensible APIs by reducing and reusing the code to accomplish data persistence. Furthermore, JPA also diminishes the disk overhead and network resource consumption because ORM maps the binding between dissimilar modeling architectures. Moreover, applications developed in JPA reduces the load of handling complex task but it is not preferred in distributed databases. The high-level abstraction, however, is seldom complex for an unfamiliar developer. But JPA reliably manages the prospective user to create an efficient API with a standardized interfaces. The prospective of the paper was to consolidate the benefits and features in the four JPA implementation i.e. Hibernate, Eclispelink, Open-JPA and Data Nucleus. A literature review on the JPAs would be advantageous for an unfamiliar developer. In con- clusion, the JPA persistence providers [30] can develop efficient API and performance oriented. But the question resides which implementation is well suited to which environment. This paper concretely established an interpretation by analyzing four distinct JPA implementation and generating a comparative summary sordid on the specialties and etiquette of the API. Hibernate being the most exceptional JPA standard in terms of the documentation support and libraries to manage complicated responsibilities. For future, an optimal solution to improve the scalability of the API through proper documentations of the complex task along with improving the cache mechanism in OpenJPA [30].

References

[1] Patrick Connor Linskey BEA Systems, Inc., San Francisco, CA Marc Prud'hommeaux BEA Systems, Inc., San Francisco, CA 2007, "An in-depth look at the architecture of an object/relational mapper SIGMOD'07," in *Proc. the 2007 ACM SIGMOD International Conference on Management of Data.*

[2] L. Sembera. (2012). Comparison of JPA providers and issues with migration. [Online]. Available: http:// is. muni.cz/ th/ 365414/fim/ thesis.pdf

[3] B. Eberlei, G. Blanco, and J. WageRoman Borschel. (2015). Doctrine of objects. [Online]. Available: http:// www.doctrine-project.org/

[4] E. E. Ogheneovo, P. O. Asagba, and N. O. Ogini, "Object relational mapping technique for java framework," *International Journal of Engineering Science Invention*, 2013.

[5] M. Keith and M. Schincariol, "Pro JPA 2, a definitive guide to mastering the java persistence API book," 2010.

[6] T. Giunipero. (2016). Developing a Java persistence API with the netbeans IDE and ecliselink. [Online]. Available: http:// www. oracle.com/technetwork/systems/ts-5400-159039.pdf

[7] O. Probst, *Investigating a Constraint-Based Approach to Data Quality in Information Systems Master Thesis*, 2013.

[8] Enoki, Y. Ozawa, H. Horii, and T. Onodera, "Memory-efficient index for cache invalidation mechanism with OPENJPA," *Web Information Systems Engineering*, 2012.

[9] D. S. Bower, "The myth: Object-relational impedance mismatch is a wicked problem exposing," *The Seventh International Conference on Advances in Databases, Knowledge, and Data Applications*, 2015.

[10] J. E. Lascano. (2014). JPA implementations versus pure JDBC. [Online]. Available: https://www.researchgate.net/publication

[11] K. L. Nitin, "JPA 2 enhancements every java developer should know," 2010.

[12] B. S. Sapre, R. V. Thakare, S. V. Kakade, and B. B. Meshram, "Design and application of the hibernate persistence layer data report system using jasper reports," *International Journal of Engineering and Innovative Technology (IJEIT)*, 2012.

[13] N. Sharma and P. N. Barwal, "Electronic project proposal management system for research projects based on integrated framework of spring and hibernate," *International Journal of Soft Computing and Engineering (IJSCE)*, 2014.

[14] B. Vasavi, Y. V. Sreevani, and G. S. Priya, "Hibernate technology for an efficient business application extension," *Journal of Global Research in Computer Science*, 2011.

[15] Eclipselink. (2015). *Eclipselink Website.* [Online]. Available: http:// www. ecli pse .org/

[16] O. Probst, "Investigating a constraint-based approach to data quality in information systems master thesis," 2013.

[17] O. Probst, "Data persistence layer and on the model-view-controller pattern for a software design case study," 2010.

[18] OPENJPA. (2013). What is enhancement anyway? *OPENJPA Website.* [Online]. Available: http://openjpa.apache.org /entityenhancement.html

[19] M. Nachev. (2013). Apache OpenJPA 2.0 user's guide. [Online]. Available: http://openjpa.apache.org/builds/2.0.0/apache-openjpa-2.0.0/docs/manual/manual.pdf

[20] J. Tee and A. Jefferson, "Slingshot yourself into datanucleus 2.1 and JPA 2.0, blog," 2014.

[21] Data Nucleus. [Online]. Available: http://www.datanu -cl eus.org/ .

[22] Oracle Contributor. (2013). EclipseLink solutions guide for eclipse link release 2.5. [Online]. Available: http://www.eclipse.org /eclipselink/documentation/2.5/eclipselinktladg.pdf

[23] H. Wu. (2014). Handling large result set. [Online]. Available: http://herbe rtwu.wordpress.com/2009/04/24handling-large-collection-data-sets.

[24] Hibernate. (2015). Multithreading in hibernate. [Online]. Available: http://blog. xebia.com/2009/02/07/hibernate-and-multi- threading

[25] Eclipselink. (2014). Eclipselink flushing. [Online]. Available: http://wiki.eclipse.org/EclipseLink/ FAQ/J PA

[26] Eclipselink. (2014). Transaction management in eclipselink. [Online]. Available: https://wiki. eclipse.org/

[27] S. Folino. (2012). JPA set vs list. [Online]. Available: http://simone-folino.blogspot.ca/2012/09/jpa-set-and-list-using-jointabml

[28] O. Yang and H. ji *et al., "*A data persistence layer model based on DAO design pattern and hibernate framework [J]," *Microcomputer Applications*, 2009.

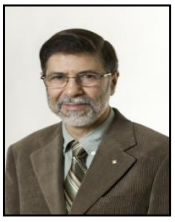
[29] C. Bauer and G. King, "Hibernate in action," 2005.

[30] Anderson, M. Kenneth, and A. Schram, "Design and implementation of a data analytics infrastructure in support of crisis informatics

research (NIER track)," in *Proc. of the 33rd International Conference on Software Engineering, ACM*, 2011.

Neha Dhingra received the B.Sc degree in computer and information science and the pursing second masters in M.Sc. (Thesis) degree from University Of Ottawa, Ottawa, Canada. She worked as a database administrator and database developer in various IT companies. Her research area is database and cloud management. She has been awarded the best paper in the Montreal conference held at University of De Quebec Montreal, 2016. She is a certified database administrator and developer with certifications in oracle and microsoft. Being a certified OCA (Oracle Certified Associate) in Oracle 12 c she has four years' IT industry experience with hand on working expertise of production environment. Currently, she is involved in the field Java persistence in cloud databases.

Emad Abdelmoghith received the M.S. degree in computer and information science and the Ph.D. degree in information technology from Cairo university, Cairo, Egypt in 2003 and 2008 respectively. His research interests include network security and software engineering. He worked as assistant professor at king Saud University, Riyadh, KSA. He has completed his research work at university of Ottawa, Ottawa, Canada.



Hussein Mouftah joined the School of Information Technology and Engineering (now School of Electrical Engineering and Computer Science) of the University of Ottawa in 2002 as a Tier 1 Canada research chair professor, where he became distinguished university professor in 2006. He has been with the ECE Dept. at Queen's University (1979-2002), where he was prior to his departure a full professor and the department associate head. He has six years' industrial experience mainly at Bell Northern Research of Ottawa (Nortel Networks). He served as editor-in-chief of the IEEE Communications Magazine (1995-97) and IEEE ComSoc director of magazines (1998-99), chair of the awards committee (2002-03), director of education (2006-07), and member of the board of governors (1997-99 and 2006-07). He has been a distinguished speaker of the IEEE communications society (2000-2007). He is the author or coauthor of 10 books, 72 book chapters and more than 1400 technical papers, 14 patents, 6 invention disclosures and 144 industrial reports. He is the joint holder of 20 best/outstanding paper awards. He has received numerous prestigious awards, such as the 2016 R.A. Fessenden Medal in telecommunications engineering of IEEE Canada, the 2015 IEEE Ottawa section outstanding educator award, the 2014 Engineering Institute of Canada K. Y. Lo Medal, the 2014 technical achievement award of the IEEE Communications Society Technical Committee on wireless Ad Hoc and sensor networks, and the Royal Society of Canada RSC Academy of Science (2008).