What is persistence?

Almost all applications require persistent data. Persistence is one of the fundamental concepts in application development. If an information system didn't preserve data when it was powered off, the system would be of little practical use. Object persistence means individual objects can outlive the application process; they can be saved to a data store and be re-created at a later point in time. When we talk about persistence in Java, we're normally talking about mapping and storing object instances in a database using SQL.

What are Relational databases?

A relational database is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

Understanding SQL

You use SQL as a ***data definition language (DDL)*** when creating, altering, and dropping artifacts such as tables and constraints in the catalog of the DBMS.

When this schema is ready, you use SQL as a ***data manipulation language (DML)*** to perform operations on data, including insertions, updates, and deletions.

You retrieve data by executing queries with restrictions, projections, and Cartesian products. For efficient reporting, you use SQL to join, aggregate, and group data as necessary. You can even nest SQL statements inside each other—a technique that uses ***subselects***.

When your business requirements change, you'll have to modify the database schema again with DDL statements after data has been stored; this is known as *schema evolution*.

Although the SQL database is one part of ORM, the other part, of course, consists of the data in your Java application that needs to be persisted to and loaded from the database.

Using SQL in Java

When you work with an SQL database in a Java application, you issue SQL statements to the database via the ***Java Database Connectivity (JDBC) API.*** Whether the SQL

was written by hand and embedded in the Java code or generated on the fly by Java code, you use the JDBC API to bind arguments when preparing query parameters, executing the query, scrolling through the query result, retrieving values from the result set, and so on.

CAP theorem.

According to this rule, a distributed system can't be **consistent**, **available**, and **tolerant against partition failures** all at the same time. A system may guarantee that all nodes will see the same data at the same time and that data read and write requests are always answered. But when a part of the system fails due to a host, network, or data center problem, you must either give up strong consistency (linearizability) or 100% availability. In practice, this means you need a strategy that detects partition failures and restores either consistency or availability to a certain degree (for example, by making some part of the system temporarily unavailable for data synchronization to occur in the background). Often it depends on the data, the user, or the operation whether strong consistency is necessary.

Object model

An object model uses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, polymorphism, and persistence. The object model enables you to create well-structured and complex systems.

In an object model system, objects are the components of the system. Objects are instances of classes, and classes are related to other classes via inheritance relationships. An object has an identity, a state, and a behavior.

An object model helps you create reusable application frameworks and systems that can evolve over time. In addition, object-oriented systems are usually smaller than non-object-oriented implementations

Relational model

A relational model defines the structure of data, data manipulation, and data integrity. Data is organized in the form of tables, and different tables are associated by means of referential integrity (a foreign key). Integrity constraints such as a primary key, unique check constraints, and not null are used to maintain an entity's integrity in the relational model.

A relational data model isn't focused on supporting entity-type inheritance: entity-based polymorphic association from an object model can't be translated into similar entities in a relational model.

In an object model, you use the state of the model to define equality between objects. But in a relational model, you use an entity's primary key to define equality of entities. Object references are used to associate different objects in an object model, whereas a foreign key is used to establish associations in a relational model. Object references in the object model facilitate easier navigation through the object graph.

Because these two models are distinctly different, you need a way to persist object entities (Java objects) into a relational database.

Object/relational mapping (ORM) frameworks help you take advantage of the features present in an object model (such as Java) and a relational model (such as database management systems [DBMSs]).

With the help of **ORM frameworks**, you can persist objects in Java to relational tables using metadata that describes the mapping between the objects and the database. The metadata shields the complexity of dealing directly with SQL and helps you develop solutions in terms of business objects.

An ORM solution can be implemented at various levels:

- **Pure relational:** An application is designed around the relational model.
- **Light object mapping:** Entities are represented as classes and are mapped manually to relational tables.
- **Medium object mapping:** An application is designed using an object model, and SQL is generated during build time using code-generation utilities.
- **Full object mapping:** Supports sophisticated object modeling including composition, inheritance, polymorphism, and persistence by reachability.

The following are the benefits of using an ORM framework:

- **Productivity:** Because you use metadata to persist and query data, development time decreases and productivity increases.
- **Prototyping:** Using an ORM framework is extremely useful for quick prototyping.
- **Maintainability:** Because much of the work is done through configuration, your code has fewer lines and requires less maintenance.
- **Vendor independence:** An ORM abstracts an application from the underlying SQL database and SQL dialect, which gives you the portability to support multiple databases. Java Specification Request 317 (JSR317) defines the Java Persistence API(JPA) specification. Using JPA means you can transparently switch between ORM frameworks such as Hibernate and TopLink.

ORM frameworks also have some disadvantages:

- **Learning curve:** You may experience a steep learning curve as you learn when and how to map and manage objects. You also have to learn a new query language.
- **Overhead:** For simple applications that use a single database and data without many business requirements for complex querying, an ORM framework can be extra overhead.
- **Slower performance:** For large batch updates, performance is slower.

ORM Frameworks

Following are the various frameworks that function on ORM mechanism: -

- Hibernate
- TopLink
- ORMLite
- iBATIS
- JPOX

Java Persistence API(JPA)

JPA are general-purpose solutions to the persistence problem that any type of Java (or Groovy, or Scala) application can use.

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

The JPA specification defines the following:

- A facility for specifying mapping metadata—how persistent classes and their properties relate to the database schema. JPA relies heavily on Java annotations in domain model classes, but you can also write mappings in XML files.
- API s for performing basic CRUD operations on instances of persistent classes, most prominently ***javax.persistence.EntityManager*** to store and load data.
- A language and API s for specifying queries that refer to classes and properties of classes. This language is the Java Persistence Query Language (JPQL) and looks similar to SQL. The standardized API allows for programmatic creation of criteria queries without string manipulation.

- How the persistence engine interacts with transactional instances to perform dirty checking, association fetching, and other optimization functions. The latest JPA specification covers some basic caching strategies.

Hibernate implements JPA and supports all the standardized mappings, queries, and programming interfaces.

JPA Versions

The first version of Java Persistence API, JPA 1.0 was released in 2006 as a part of EJB 3.0 specification.

Following are the other development versions released under JPA specification

- JPA 2.0 - Following are the important features of this version: -
  - It supports validation.
  - It expands the functionality of object-relational mapping.
  - It shares the object of cache support.
- JPA 2.1 - Following features: -
  - It allows fetching of objects.
  - It provides support for criteria update/delete.
  - It generates schema.
- JPA 2.2 - Some of its important features are: -
  - It supports Java 8 Date and Time.
  - It provides @Repeatable annotation that can be used when we want to apply the same annotations to a declaration or type use.
  - It allows JPA annotation to be used in meta-annotations.
  - It provides an ability to stream a query result.

Java Persistence consists of four areas: -

- The Java Persistence API
- The query language
- The Java Persistence Criteria API
- Object/relational mapping metadata

Summary

- With object persistence, individual objects can outlive their application process, be saved to a data store, and be re-created later. The object/relational mismatch comes into play when the data store is an SQL -based relational database management system. For instance, a network of objects can’t be saved to a database table; it must be disassembled and persisted to columns of portable SQL data types. A good solution for this problem is object/relational mapping (ORM).
- ORM isn’t a silver bullet for all persistence tasks; its job is to relieve the developer of 95% of object persistence work, such as writing complex SQL statements with many table joins and copying values from JDBC result sets to objects or graphs of objects.
- A full-featured ORM middleware solution may provide database portability, certain optimization techniques like caching, and other viable functions that aren’t easy to hand-code in a limited time with SQL and JDBC.