1. upload pdf document

2. Vendor Login -> Listing (showcontact) -> List (viewdocument) - showdocuments

JDBC - >RDBMS

JPA query agnotsitc

JDBC -> Queries DB Specific

Data JPA and MyBatis

<https://www.baeldung.com/jpa-vs-jdbc>

https://medium.com/an-idea/spring-boot-spring-data-jpa-vs-mybatis-514d969648ee

The dialect specifies the type of database used in hibernate so that hibernate generate appropriate type of SQL statements. For connecting any hibernate application with the database, it is required to provide the configuration of SQL dialect.

Main difference between JPA and JDBC is level of abstraction.

JDBC is a low level standard for interaction with databases. JPA is higher level standard for the same purpose. JPA allows you to use an object model in your application which can make your life much easier. JDBC allows you to do more things with the Database directly, but it requires more attention. Some tasks can not be solved efficiently using JPA, but may be solved more efficiently with JDBC.

Why use JPA instead of directly writing SQL query on Java File (i.e. directly to JDBC) ?

Certain projects require engineers to focus more on the object model rather than on the actual SQL queries used to access data stores. The question can actually be interpreted as

Why should one use an ORM framework ?

which can have different answers in different contexts.

Most projects can benefit from having a domain model, with persistence being a second concern. With JPA (implementations) or most other ORM frameworks, it is possible to have all entities i.e. tables in your database, modelled as classes in Java. Additionally, it also possible to embed behavior into these classes and therefore achieve a behaviorally rich domain model. The entities in this model can have multiple purposes, including the purpose of replacing DTOs for transporting data across tiers.

That said, there are places where ORM frameworks may not be a direct fit to the problem, especially when the data model is already established, or when one is working with legacy systems where mapping database tables to Java classes is a non-trivial exercise. And in certain cases, if one needs to absolutely tune the heck out of the SQL generated by the ORM framework, then ORM frameworks are usually a bad fit.

Various caching strategies are still the way to scale JPA/Hibernate (you basically named the most popular options in your question). Nothing extraordinary happend since 4-5 years in this field, as far as I know. One more option you haven't mentioned is JBoss Cache. So the Second Level Cache for JPA/Hibernate still rules in this area.

Why no progress here? My wild guess is that first of all people, who need scalable application tend to ignore JPA and Hibernate in areas where high performance is needed. Usually people go with SQL dressed in Spring Framework JDBCTemplate helpers and transaction management. Then scalability is the matter of database capabilities in this area.

The other trend is to use No-SQL databases. There is plany of solutions: MongoDB, CouchoDB, Cassandra, Redis, to name a few. These are usually Google BigTable like key-value storages (this is oversimplification, but it is more or less the idea behind that approach) and they scale as hell, if you accept their limitations (relations are no longer managed easily, etc.).

|  |  |
| --- | --- |
| JDBC | JPA |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table

Description automatically generated