What is Java Persistence API (JPA)?

Java Persistence API (JPA) provides specification for managing the relational data in applications. Current JPA version 2.1 was started in July 2011 as JSR 338. JPA 2.1 was approved as final on 22 May 2013.

JPA specifications is defined with annotations in javax.persistence package. Using JPA annotation helps us in writing implementation independent code.

What are the important benefits of using Hibernate Framework?

Some of the important benefits of using hibernate framework are:

Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business logic.

Hibernate framework provides support for XML as well as JPA annotations, that makes our code implementation independent.

Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.

Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.

Hibernate is easy to integrate with other Java EE frameworks, it's so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.

Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it's required.

Hibernate cache helps us in getting better performance.

For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

Overall hibernate is the best choice in current market for ORM tool, it contains all the features that you will ever need in an ORM tool.

What are the advantages of Hibernate over JDBC?

Some of the important advantages of Hibernate framework over JDBC are:

Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks more cleaner and readable.

Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.

Hibernate implicitly provides transaction management, in fact most of the queries can't be executed outside transaction. In JDBC API, we need to write code for transaction management using commit and rollback. Read more at JDBC Transaction Management.

JDBC API throws SQLException that is a checked exception, so we need to write a lot of try-catch block code. Most of the times it's redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throw JDBCException or HibernateException un-checked exception, so we don't need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.

Hibernate Query Language (HQL) is more object oriented and close to java programming language. For JDBC, we need to write native sql queries.

Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.

Hibernate provide option through which we can create database tables too, for JDBC tables must exist in the database.

Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection pool. This is very important feature in enterprise application and completely missing in JDBC API.

Hibernate supports JPA annotations, so code is independent of implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

Name some important interfaces of Hibernate framework?

Some of the important interfaces of Hibernate framework are:

SessionFactory (org.hibernate.SessionFactory): SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We need to initialize SessionFactory once and then we can cache and reuse it. SessionFactory instance is used to get the Session objects for database operations.

Session (org.hibernate.Session): Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction. We should open session only when it's required and close it as soon as we are done using it. Session object is the interface between java application code and hibernate framework and provide methods for CRUD operations.

Transaction (org.hibernate.Transaction): Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.

Query(org.hibernate.Query)
Query objects use the SQL or Hibernate Query Language (HQL) string to recover data from the database and generate objects. A Query request is used to bind inquiry parameters, bound the number of results given back by the query, and lastly to perform the query.

What is hibernate configuration file?

Hibernate configuration file contains database specific configurations and used to initialize SessionFactory. We provide database credentials or JNDI resource information in the hibernate configuration xml file. Some other important parts of hibernate configuration file is Dialect information, so that hibernate knows the database type and mapping file or class details.

What is hibernate mapping file?

Hibernate mapping file is used to define the entity bean fields and database table column mappings. We know that JPA annotations can be used for mapping but sometimes XML mapping file comes handy when we are using third party classes and we can't use annotations.

Name some important annotations used for Hibernate mapping?

Hibernate supports JPA annotations and it has some other annotations in org.hibernate.annotations package. Some of the important JPA and hibernate annotations used are:

javax.persistence.Entity: Used with model classes to specify that they are entity beans.

javax.persistence.Table: Used with entity beans to define the corresponding table name in database.

javax.persistence.Access: Used to define the access type, either field or property. Default value is field and if you want hibernate to use getter/setter methods then you need to set it to property.

javax.persistence.Id: Used to define the primary key in the entity bean.

javax.persistence.EmbeddedId: Used to define composite primary key in the entity bean.

javax.persistence.Column: Used to define the column name in database table.

javax.persistence.GeneratedValue: Used to define the strategy to be used for generation of primary key. Used in conjunction with javax.persistence.GenerationType enum.

javax.persistence.OneToOne: Used to define the one-to-one mapping between two entity beans. We have other similar annotations as OneToMany, ManyToOne and ManyToMany

org.hibernate.annotations.Cascade: Used to define the cascading between two entity beans, used with mappings. It works in conjunction with org.hibernate.annotations.CascadeType

javax.persistence.PrimaryKeyJoinColumn: Used to define the property for foreign key. Used with org.hibernate.annotations.GenericGenerator and org.hibernate.annotations.Parameter

Here are two classes showing usage of these annotations.

```
mport javax.persistence.Access;
```

```
import javax.persistence.AccessType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import org.hibernate.annotations.Cascade;

@Entity
@Table(name = "EMPLOYEE")
@Access(value=AccessType.FIELD)
public class Employee {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "emp_id")
	private long id;

	@Column(name = "emp_name")
	private String name;

	@OneToOne(mappedBy = "employee")
	@Cascade(value = org.hibernate.annotations.CascadeType.ALL)
	private Address address;

	//getter setter methods
}
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name = "ADDRESS")
@Access(value=AccessType.FIELD)
public class Address {

	@Id
	@Column(name = "emp_id", unique = true, nullable = false)
	@GeneratedValue(generator = "gen")
	@GenericGenerator(name = "gen", strategy = "foreign", parameters = {
@Parameter(name = "property", value = "employee") })
	private long id;

	@Column(name = "address_line1")
	private String addressLine1;

	@OneToOne
	@PrimaryKeyJoinColumn
	private Employee employee;

	//getter setter methods
}
```

What is Hibernate SessionFactory and how to configure it?

SessionFactory is the factory class used to get the Session objects. SessionFactory is responsible to read the hibernate configuration parameters and connect to the database and provide Session objects. Usually an application has a single SessionFactory instance and threads servicing client requests obtain Session instances from this factory.

The internal state of a SessionFactory is immutable. Once it is created this internal state is set. This internal state includes all of the metadata about Object/Relational Mapping.

SessionFactory also provide methods to get the Class metadata and Statistics instance to get the stats of query executions, second level cache details etc.

Hibernate SessionFactory is thread safe?

Internal state of SessionFactory is immutable, so it's thread safe. Multiple threads can access it simultaneously to get Session instances.

What is Hibernate Session and how to get it?

Hibernate Session is the interface between java application layer and hibernate. This is the core interface used to perform database operations. Lifecycle of a session is bound by the beginning and end of a transaction.

Session provide methods to perform create, read, update and delete operations for a persistent object. We can execute HQL queries, SQL native queries and create criteria using Session object.

Hibernate Session is thread safe?

Hibernate Session object is not thread safe, every thread should get it's own session instance and close it after it's work is finished.

What are different states of an entity bean or Hibernate State?

An element bean occurrence can exist is one of the three states.
Transient: When a object is never continued or related with any session, it's in transient state. Transient cases might be influenced diligent by calling **save()**, **persist()** or **saveOrUpdate()**. Steady examples might be influenced transient by calling to **delete().**
Persistent: When a question is related with a one of a kind session, it's in persevering state. Any occurrence returned by a get() or load() technique is persevering.
Detached: When a object is already relentless yet not related with any session, it's in disengaged state. Disengaged occasions might be made determined by calling **refresh()**, **saveOrUpdate()**, **lock()** or **replicate()**. The condition of a transient or disengaged example may likewise be influenced relentless as another constant case by calling to **merge()**.

What is the purpose of Session.beginTransaction()?

In Hibernate, every exchange of data is maintained using a transaction. Whenever a new data exchange is to be started, the function **Session.beginTransaction** is called to start the transaction.

What can be done to completed a transaction in Hibernate?
In order to complete a transaction in hibernate, there can be two actions:

1. Commit
2. Rollback

Whenever a transaction is committed, the data exchange is written to the database. In case of rollback, the transaction data is flushed and never written or updated to the database.

How do we commit a transaction in Hibernate?

The transaction object in Hibernate can either be committed or rolled back. In order to perform this action, we utilise the below code.

```
1    tx = Session.beginTransaction();
2    ...
3    ...
4    ...
5    //Do something with transaction
6    ...
7    ...
8    ...
9    tx.commit();
```

As it can be seen, the function call `tx.commit()` does the task of committing the transaction to the database. For a rollback, the procedure remains the same. All you need to do is change the function call to `tx.rollback()`.

Which are the different types of relationships available in Hibernate mapping?

There are three different types of relationships that can be implemented in hibernate. They are:

1. One-to-One mapping
2. One-to-Many mapping
3. Many-to-Many mapping

Which annotation is used to define the mapping in Java files?

In order to define mapping in Java files, there are three different annotations that can be used as needed. The annotations are:

1. `@OneToOne`
2. `@ManyToOne`
3. `@OneToMany`

The usage of the annotations is quite clear. The annotation requires a mandatory attribute `mappedBy` to identify the related table.

. Is it possible to connect multiple database in a single Java application using Hibernate?

Yes. Practically it is possible to connect a single Java application to multiple databases using two separate hibernate configuration files and two separate session

factories. These hibernate configuration files contain different dialect configuration pertaining to the respective database. The entities are exclusively mapped to the relevant database configuration. Thus, with two different parallel SessionFactory objects, it is possible to have multiple databases connected.

What is a Criteria query in Hibernate?

A criteria query in Hibernate is a complex hibernate query that allows the developer to specify multiple conditions to fetch the data from database using a Hibernate query in the backend. A criteria query is created using a `Criteria` object. This criteria object is later combined with restrictions to get the final result. A simple criteria query execution is demonstrated in the code below:

```
1    Criteria cr = session.createCriteria(User.class);
2    cr.add(Restrictions.eq("user_name", datsabk));
3    List results = cr.list();
```

The above query fetches the list of users with the username attribute having value **datsabk**. Thus, Criteria class provides the necessary implementation to simplify complex queries using objects.

What is lazy loading in Hibernate?

it is possible to map the entities with other entities using annotations in Hibernate. When you fetch such entities, hibernate is able to load the associated entities in two ways. One of them is to load them along with the parent entity. This often is time consuming due to the joins and multiplications of records involved.
The other way of loading is called the Lazy Loading way. Lazy loading means that Hibernate would load the associated entity only when you actually try to access the associated entity values. This not only saves you runtime memory but also speeds up the query.

What is load() method of session Object?

This method is used to load a proxy object without actually hitting the database. Thus, it saves you the hit to the database and loads the data from the cache. This is an explicit way to tell hibernate to pick the data from cache.

Does Hibernate support polymorphism?

Yes. Hibernate inherently supports polymorphism. Hibernate classes mainly support its operations via polymorphism itself.

How many Hibernate sessions exist at any point of time in an application?
Hibernate session is a shared object. During any point of time, there exists only single shared session object that helps in managing transactions and getting connections from the connection pool. It must be noted that this is true only when a single database configuration is used. In case of multiple database configurations, Hibernate would create a separate session object to maintain the mapping and transactions for the other database.

Hibernate Caching and implementation

Caching is an important feature that is required for enterprise level applications handling thousands of requests daily. Hibernate framework inherently supposed caching of queries outputs at various levels. These caching levels are configurable to best suite your needs. There are three types of caching levels:

1. **Session Level caching:** Session level caching is implemented by Hibernate as a default caching level. This caching features caching of objects in the session. These cached objects reside in the session memory itself.
2. **Second Level cache:** The second level cache is responsible to cache objects across sessions. When the second level caching is enabled, the objects are first searched in the cache and later if not found, a database query is executed to search for it. Second level cache will be used whenever the objects are being loaded using their primary key. Second level cache objects also fetch their respective associations and reside in a separate memory stack other than session.
3. **Query Cache:** Question Cache is utilized to store the after-effects of a query. At the point when the query caching is turned on, the outputs of the query are put away against the fired query and parameters. Each time the query is let go the store framework checks for the mix of parameters and query. In the event that the outcomes are found in the cache, they are returned, generally a database transaction is started. As should be obvious, it's anything but a smart thought to reserve an inquiry in the event that it has various parameters, since then a solitary parameter can take various qualities. For every one of these blends the outcomes are put away in the memory. This can result in larger memory utilization.

How to configure the database URL and credentials in hibernate.cfg.xml?

Hibernate framework can be configured using variety of property values. In the above questions, we discussed how to configure the framework using property file. The configuration in XML is quite similar. We just need to create the respective property tags with the names similar to the ones specified in the properties file above. The example of configuration database URL and credentials is provided below.

```
<property name = "hibernate.connection.url">
  jdbc:mysql://localhost/mydb
</property>

<property name = "hibernate.connection.username">
  root
</property>

<property name = "hibernate.connection.password">
  password
</property>
```

How do you configure the dialect in hibernate.cfg.xml?

The configuration of dialect in xml involves defining of a property with the name **hibernate.dialect.** A sample XML tag for defining the dialect is shown below:

```
<property name="org.hibernate.dialect.MySQLDialect">
org.hibernate.dialect.MySQLDialect
</property>
```

How do we define the primary key value generation logic?

The primary key values can be generated using various methods depending on the database. For instance, in MySQL database the primary keys can be generated using an auto-incrementing algorithm while in Oracle database, you need a sequence to be created and used for an auto-incrementing the value for primary key. These methods of generation can be specified using the below annotation code.

```
@Entity
@Table(name="users")
public class User{

@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
int userid;

@Column(name="user_name")
String username;

String password;
}
```

The userid column here has been defined as a primary key autogenerated using the identity strategy. The possible values for `strategy` include:

- GenerationType.AUTO
- GenerationType.IDENTITY
- GenerationType.SEQUENCE
- GenerationType.TABLE

. How do we specify a variable to be primary key for the table?

Hibernate is able to create the database tables for an application directly based on the mappings that are provided in the Java code. In such a scenario, Hibernate requires to know which columns are expected to be primary keys. This can be configured using the annotation `@Id`. Hibernate not only takes care of creating this

column as primary key column but also validates its unique constraint during every database insertion and update.

How do we specify a different column name for the variables mapping?

The annotation `@Column` is used to define a column name associated with a variable. In absence of this annotation, Hibernate precompiles the mapping of variable mapped to the column with same name. An example of using this annotation is shown below:

```
@Entity
@Table(name="users")
public class User{
@Column(name="user_name")
String username;
String password;
}
```

How do I specify table name linked to an entity using annotation?

As it can be seen in the above code, the annotation `@Table` is used to specify the database table name linked to the entity.This entity requires a mandatory attribute `name` which specifies the table name as in the database.

Which annotation is used to declare a class as a hibernate bean?

The annotation `@Entity` is used to declare a class as an entity. A simple example is shown below.

```
@Entity
@Table(name="users")
public class User{
String username;
String password;
}
```

What are the configurations involved in Hibernate framework?

Hibernate framework is an enormous framework designed to handle almost every database operation for you. To configure such a framework involves multiple entities to be configured. These include:

- Database credentials
- Database dialect
- Database URL
- Caching levels
- ORM mapping
- Connection pool configuration
- Table creation mode – Create/Update

Hibernate comes with default values for almost every non-database configuration like the Connection pool size, caching levels, table creation mode and others. Thus, despite having so many configurable aspects, it allows to get started with minimal configuration.

What are different states of an entity bean?

An entity bean instance can exist is one of the three states.

1. **Transient**: When an object is never persisted or associated with any session, it's in transient state. Transient instances may be made persistent by calling save(), persist() or saveOrUpdate(). Persistent instances may be made transient by calling delete().
2. **Persistent**: When an object is associated with a unique session, it's in persistent state. Any instance returned by a get() or load() method is persistent.
3. **Detached**: When an object is previously persistent but not associated with any session, it's in detached state. Detached instances may be made persistent by calling update(), saveOrUpdate(), lock() or replicate(). The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().

What is cascading and what are different types of cascading?

When we have relationship between entities, then we need to define how the different operations will affect the other entity. This is done by cascading and there are different types of it.

Here is a simple example of applying cascading between primary and secondary entities.

```
import org.hibernate.annotations.Cascade;

@Entity
```

```
@Table(name = "EMPLOYEE")
public class Employee {

@OneToOne(mappedBy = "employee")
@Cascade(value = org.hibernate.annotations.CascadeType.ALL)
private Address address;

}
```

Note that Hibernate CascadeType enum constants are little bit different from JPA javax.persistence.CascadeType, so we need to use the Hibernate CascadeType and Cascade annotations for mappings, as shown in above example.
Commonly used cascading types as defined in CascadeType enum are:

1. None: No Cascading, it's not a type but when we don't define any cascading then no operations in parent affects the child.
2. ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. Basically everything
3. SAVE_UPDATE: Cascades save and update, available only in hibernate.
4. DELETE: Corresponds to the Hibernate native DELETE action, only in hibernate.
5. DETATCH, MERGE, PERSIST, REFRESH and REMOVE – for similar operations
6. LOCK: Corresponds to the Hibernate native LOCK action.
7. REPLICATE: Corresponds to the Hibernate native REPLICATE action.

What is the difference between save() and persist() method in Hibernate?

The foremost difference between save() and persist() system is that save returns the result of a Serializable object while the return type of persisting () system is void, so it doesn't yield anything as such.

What is first level cache in Hibernate?

It is session reserve and compulsory cache. It is from the first level store through which every one of the solicitations must pass.

The session question stores an object under its control before submitting it to the database.

Q29). What is the second level cache in Hibernate?

It is a discretionary cache. What's more, dependably the primary level reserve will be counseled before any endeavor is performed to find an object in the second level store. This store can be designed on a pre-accumulation and per-class premise and it is for the most part in charge of reserving objects over the sessions.

Q30). What is Query level cache in Hibernate?.

In sleep, a store question can be actualized that outcomes in sets and incorporates intimately with the second level cache. It is a discretionary component and it requires two extra reserve locales that can hold the stored inquiry results and furthermore the timestamps at whatever point a table is refreshed. This is helpful just for the inquiries that run every now and again holding similar parameters.

How to integrate Hibernate and Spring frameworks?

Spring is one of the most used Java EE Framework and Hibernate is the most popular ORM framework. That's why Spring Hibernate combination is used a lot in enterprise applications. The best part with using Spring is that it provides out-of-box integration support for Hibernate with **Spring ORM**module. Following steps are required to integrate Spring and Hibernate frameworks together.

1. Add hibernate-entitymanager, hibernate-core and spring-orm dependencies.
2. Create Model classes and corresponding DAO implementations for database operations. Note that DAO classes will use SessionFactory that will be injected by Spring Bean configuration.
3. If you are using Hibernate 3, you need to configure org.springframework.orm.hibernate3.LocalSessionFactoryBean or org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean in Spring Bean configuration file. For Hibernate 4, there is single class org.springframework.orm.hibernate4.LocalSessionFactoryBean that should be configured.
4. Note that we don't need to use Hibernate Transaction Management, we can leave it to Spring declarative transaction management using @Transactional annotation.