Explain dynamic insert and dynamic update properties in Hibernate

Answer

dynamicinsert and dynamic update generates hibernate insert and update statement based on properties are set or not , and updated or not..

Hibernate creates SQL statements for CRUD operations for each of its persistent classes during application startup and cache them in memory.

The cached update statement will update every field even if the value isn’t changed; it is simply updated with the old value.

Turning off this feature may improve the startup time for the application.

You can turn off this feature using dynamic-insert and dynamic-update attributes of the class element in a mapping file or using @DynamicInsert,

with @DynamicUpdate annotations at the class level.

With the dynamic-insert property set to true, Hibernate does not include null values for properties that are not set by the application,

during an INSERT. With the dynamic-update property set to true, Hibernate does not include unmodified properties in the UPDATE.

Sample Mapping file snippet

<hibernate-mapping package="com.javajee.hibexamples" auto-import="false">

<class name="Sample" table="SAMPLE" dynamic-insert="true" dynamic-update="true">

…

//////cant use not stattic member in static method

</hibernate-mapping>

//-------------------------------------------------------------------------------------------------------------------------------

How to use DynamicUpdate and SelectBeforeUpdate in Hibernate

If you are using openSession() then you have to use both DynamicUpdate and SelectBeforeUpdate to make it effective.

As openSession every time opens new session, any object you want to update, that object doesn't lies in that session so you have to use SelectBeforeUpdate

to retrieve that object in session. Then and then hibernate can determine how many fields are actually changed.

So hibernate will update only changed field as we have set DynamicUpdate=true.

If you are using annotation then syntax should be like

@Entity

@Table(name = "Abc", catalog = "xyz")

@org.hibernate.annotations.Entity(dynamicUpdate = true, selectBeforeUpdate = true)

public class Abc implements java.io.Serializable {

...

...

}

If you are using latest jar files then dynamicUpdate and selectBeforeUpdate are deprecated.

For that syntax should be like

@Entity

@Table(name = "Abc", catalog = "xyz")

@DynamicUpdate

@SelectBeforeUpdate

public class Abc implements java.io.Serializable {

...

...

}

And if you are using getCurrentSession() then no need to use SelectBeforeUpdate as object is already lies in session.

I recommend that use openSession() method instead getCurrentSession() because getCurrentSession automatically flush the session as well as close the session.

It may chance to face problem in getCurrentSession method. While in openSession we have to manage all the things.

//-------------------------------------------------------------------------------------------------------------------------------

DIFFERENT SAVING OPTION IN Hibernate

ny entity instance in your application appears in one of the three main states in relation to the Session persistence context:

transient — this instance is not, and never was, attached to a Session; this instance has no corresponding rows in the database; it’s usually just a new object that you have created to save to the database;

persistent — this instance is associated with a unique Session object; upon flushing the Session to the database, this entity is guaranteed to have a corresponding

consistent record in the database;

detached — this instance was once attached to a Session (in a persistent state), but now it’s not; an instance enters this state if you evict it from the context, clear or close the Session,

or put the instance through serialization/deserialization process.

\_\_>>>>>3.1. Persist

The persist method is intended for adding a new entity instance to the persistence context, i.e. transitioning an instance from transient to persistent state.

We usually call it when we want to add a record to the database (persist an entity instance):

1

2

3

Person person = new Person();

person.setName("John");

session.persist(person);

What happens after the persist method is called? The person object has transitioned from transient to persistent state.

The object is in the persistence context now, but not yet saved to the database. The generation of INSERT statements will occur only upon commiting the transaction,

flushing or closing the session.

Notice that the persist method has void return type. It operates on the passed object “in place”, changing its state. The person variable references the actual persisted object.

This method is a later addition to the Session interface. The main differentiating feature of this method is that it conforms to the

JSR-220 specification (EJB persistence). The semantics of this method is strictly defined in the specification, which basically states, that:

a transient instance becomes persistent (and the operation cascades to all of its relations with cascade=PERSIST or cascade=ALL),

if an instance is already persistent, then this call has no effect for this particular instance (but it still cascades to its relations with cascade=PERSIST or cascade=ALL),

if an instance is detached, you should expect an exception, either upon calling this method, or upon committing or flushing the session.

Notice that there is nothing here that concerns the identifier of an instance. The spec does not state that the id will be generated right away,

regardless of the id generation strategy. The specification for the persist method allows the implementation to issue

statements for generating id on commit or flush, and the id is not guaranteed to be non-null after calling this method, so you should not rely upon it.

You may call this method on an already persistent instance, and nothing happens. But if you try to persist a detached instance, the implementation is bound to throw an exception. In the following example we persist the entity, evict it from the context so it becomes detached, and then try to persist again. The second call to session.persist() causes an exception, so the following code will not work:

Person person = new Person();

person.setName("John");

session.persist(person);

session.evict(person);

session.persist(person); // PersistenceException!

\_\_\_\_\_\_\_>>>>>>>3.2. Save

The save method is an “original” Hibernate method that does not conform to the JPA specification.

Its purpose is basically the same as persist, but it has different implementation details.

The documentation for this method strictly states that it persists the instance, “first assigning a generated identifier”. The method is guaranteed to return the Serializable value of this identifier.

Person person = new Person();

person.setName("John");

Long id = (Long) session.save(person);

The effect of saving an already persisted instance is the same as with persist. Difference comes when you try to save a detached instance:

Person person = new Person();

person.setName("John");

Long id1 = (Long) session.save(person);

session.evict(person);

Long id2 = (Long) session.save(person);

The id2 variable will differ from id1. The call of save on a detached instance creates a new persistent instance and assigns it

a new identifier, which results in a duplicate record in a database upon committing or flushing.

\_\_\_\_\_\_\_>>>>>>>

3.3. Merge

The main intention of the merge method is to update a persistent entity instance with new field values from a detached entity instance.

For instance, suppose you have a RESTful interface with a method for retrieving an JSON-serialized object by its id to the calle

r and a method that receives an updated version of this object from the caller.

An entity that passed through such serialization/deserialization will appear in a detached state.

After deserializing this entity instance, you need to get a persistent entity instance from a persistence context and update its fields with new values from this detached instance.

So the merge method does exactly that:

finds an entity instance by id taken from the passed object (either an existing entity instance from the persistence context is retrieved,

or a new instance loaded from the database);

copies fields from the passed object to this instance;

returns newly updated instance.

In the following example we evict (detach) the saved entity from context, change the name field, and then merge the detached entity.

Person person = new Person();

person.setName("John");

session.save(person);

session.evict(person);

person.setName("Mary");

Person mergedPerson = (Person) session.merge(person);

Note that the merge method returns an object — it is the mergedPerson object that was loaded into persistence context and updated, not the person object that you passed as an argument. Those are two different objects, and the person object usually needs to be discarded (anyway, don’t count on it being attached to persistence context).

As with persist method, the merge method is specified by JSR-220 to have certain semantics that you can rely upon:

if the entity is detached, it is copied upon an existing persistent entity;

if the entity is transient, it is copied upon a newly created persistent entity;

this operation cascades for all relations with cascade=MERGE or cascade=ALL mapping;

if the entity is persistent, then this method call does not have effect on it (but the cascading still takes place).

\_\_\_\_\_\_\_>>>>>>>

Update

As with persist and save, the update method is an “original” Hibernate method that was present long before the merge method was added. Its semantics differs in several key points:

it acts upon passed object (its return type is void); the update method transitions the passed object from detached to persistent state;

this method throws an exception if you pass it a transient entity.

In the following example we save the object, then evict (detach) it from the context, then change its name and call update.

Notice that we don’t put the result of the update operation in a separate variable, because the update takes place on the person object itself.

Basically we’re reattaching the existing entity instance to the persistence context — something the JPA specification does not allow us to do.

Person person = new Person();

person.setName("John");

session.save(person);

session.evict(person);

person.setName("Mary");

session.update(person);

Trying to call update on a transient instance will result in an exception. The following will not work:

Person person = new Person();

person.setName("John");

session.update(person); // PersistenceException!

\_\_\_\_\_\_\_>>>>>>>

SaveOrUpdate

This method appears only in the Hibernate API and does not have its standardized counterpart.

Similar to update, it also may be used for reattaching instances.

Actually, the internal DefaultUpdateEventListener class that processes the update method is a subclass of DefaultSaveOrUpdateListener,

just overriding some functionality. The main difference of saveOrUpdate method is that it does not throw exception when applied to

a transient instance; instead, it makes this transient instance persistent. The following code will persist a newly created instance of Person:

Person person = new Person();

person.setName("John");

session.saveOrUpdate(person);

You may think of this method as a universal tool for making an object persistent

regardless of its state wether it is transient or detached.

What to Use?

If you don’t have any special requirements, as a rule of thumb,

you should stick to the persist and merge methods, because they are standardized and guaranteed to conform to the JPA specification.

They are also portable in case you decide to switch to another persistence provider,

but they may sometimes appear not so useful as the “original” Hibernate methods, save, update and saveOrUpdate.

//--------------------------------------------------------------------------------------------------------------------------------------------------------

What is hibernate and advantage of it?

HIbenrate simplifies the application and database connection by mapping your java object with corresponsig

row of database table

->>Open source & ORM ,database independent

->>high performance

->>hql

//--------------------------------------------------------------------------------------------------------------------------------------------------------

Hibernat is orm implementation, OBject relational mapping that describes mapping of object with relational database using metada that can be xml config or annotation

mapping describes mapping between object to rdbms column

//--------------------------------------------------------------------------------------------------------------------------------------------------------

Diference between get() and load() hibernate..

difference between get() vs load method is that get() involves database hit

if object doesn't exists in Session Cache and returns a fully initialized

object which may involve several database call while load() method can return

proxy in place and only initialize the object or hit the database if any method

other than getId() is called on persistent or entity object.

This lazy initialization can save couple of database round-trip which result

in better performance.

1. Behavior when Object is not found in Session Cache

Apart from performance this is another difference between get and load which is worth remembering. get method of Hibernate Session class returns null

if object is not found in cache as well as on database while

load() method throws ObjectNotFoundException if object is not found on cache as well as on database but never return null.

2. Database hit

Get method always hit database while load() method may not always hit the database, depending upon which method is called.

3. Proxy

Get method never returns a proxy, it either returns null or fully initialized Object,

while load() method may return proxy, which is the object with ID but without initializing other properties,

which is lazily initialized. If you are just using returned object for creating relationship and only need Id then load() is the way to go.

4. Performance

By far most important difference between get and load in my opinion. get method will return a completely initialized object if

Object is not on the cache but exists on Database, which may involve multiple round-trips to database based upon object relational mappings

while load() method of Hibernate can return a proxy which can be initialized on demand

(lazy initialization) when a non identifier method is accessed.

Due to above reason use of load method will result in slightly better performance,

but there is a caveat that proxy object will throw ObjectNotFoundException later if corresponding row doesn’t exists in database,

instead of failing immediately so not a fail fast behavior.

5. load method exists prior to get method which is added on user request.

========================================================================================================================================================================================================

save vs persist

1)First difference between save and persist is there return type. Similar to save method persist also INSERT records

into database but return type of persist is void while return type of save is Serializable object.

2) Another difference between persist and save is that both methods make a transient instance persistent.

However, persist() method doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time.

Read more: http://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html#ixzz54qdFOiD2

3) One more thing which differentiate persist and save method in Hibernate is that is there behavior on outside of transaction boundaries.

persist() method guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries.

save() method does not guarantee the same, it returns an identifier, and if an INSERT has to be executed to get the identifier

(e.g. "identity" generator), this INSERT happens immediately, no matter if you are inside or outside of a transaction.

4) Fourth difference between save and persist method in Hibernate is related to previous difference on save vs persist

. Because of its above behavior of persist method outside transaction boundary,

its useful in long-running conversations with an extended Session context.

On the other hand save method is not good in a long-running conversation with an extended Session context

Read more: http://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html#ixzz54qd53yY6

==============================================================================================================================================

What is the difference between sorted and ordered collection in hibernate?

This is one of the easy Hibernate interview questions you ever face.

A sorted collection is sorted in memory by using Java Comparator

while an ordered collection uses database's order by clause

for ordering. For large data set it's better to use ordered collection to avoid any OutOfMemoryError in Java, by trying to sort them in memory.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qhTJkJr

============================================================================================================================================================================================================

What is Session in Hibernate? Can we share single Session among multiple threads in Hibernate?

This is usually asked as follow-up question of previous Hibernate Interview question.

After SessionFactory its time for Session. Session represent a small unit of work in Hibernate,

they maintain a connection with the database and they are not thread-safe,

it means you can not share Hibernate Session between multiple threads.

Though Session obtains database connection lazily it's good to close session as soon as you are done with it.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qhc1e3C

=================================================================================================================================================================

What is SessionFactory in Hibernate? is SessionFactory thread-safe?

Another common Interview questions related to Hibernate framework. SessionFactory, as the name suggest, is a factory to hibernate

Session objects. SessionFactory is often built during start-up and used by application

cation has just one SessionFactory, and individual threads

, which are servicing client’s request obtain hibernate Session instances

from this factory, that’s why any implementation of SessionFactory interface must be thread-safe.

Also, the internal state of SessionFactory, which contains all metadata about Object/Relational mapping is Immutable and can not be changed once created.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qhh0MGX

=========================================================================================================================================

n Hibernate, Object can remain in three state transient, persistent or detached.

An object which is associated with Hibernate session is called persistent object. Any change in this object will reflect in the database based on your flush strategy

i.e. automatic flush whenever any property of object change or explicit flushing by calling Session.flush() method.

On the other hand, if an object which is earlier associated with Session, but currently not associated with it are called detached object.

You can reattach detached object to any other session by calling either update() or saveOrUpdate() method on that session. T

ransient objects have newly created an instance of persistence class,

which is never associated with any Hibernate Session. Similarly, you can call persist() or save() methods to make a transient object persistent.

Just remember, here transient doesn’t represent transient keyword in Java, which is an altogether different thing.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qiHEDAE

==================================================================================================================================================================================================================================================================================

Difference between First and Second Level Cache in Hibernate

If you have used Hibernate in past then you know that one of the strongest points of Hibernate framework is caching,

which can drastically improve the performance of Java application's persistence layer if configured and used correctly.

Hibernate provides caching at many levels e.g. first level cache at Session level, second level cache at the SessionFactory level,

and query cache to cache frequently executed SQL queries. The first level cache minimizes database access for the same object.

For example, if you call the get() method to access Employee object with id = 1 from one session, it will go the database and

load the object into memory, but it will also cache the object in the first level cache.

When you will call the get() method again for the same object from the same session, even after doing some updates on the object,

it will return the object from the cache without accessing the database. You can confirm this from Hibernate's log file by observing

how many queries are executed. This is also one of the frequently asked Hibernate Interview Questions, so it will not only help to

improve the performance of your Java application but also help you to do well on your next interview.

This session level cache greatly improves the performance of Java application by minimizing database roundtrips and executing

less number of queries. For example, if an object is modified several times within the same transaction, then Hibernate will

only generate one SQL UPDATE statement at the end of the transaction, containing all the modification.

But, since this cache is associated with the Session object, which is a short-lived object in Hibernate, as soon as

you close the session, all the information held in the cache is lost. So, if you try to load the same object using the get()

method, Hibernate will go to the database again and fetch the record.

This poses significant performance challenge in an application where multiple sessions are used, but you don't need to worry.

Hibernate provides another application level cache, known as second level cache, which can be shared among multiple sessions.

This means a request for the same object will not go to the database even if it is executed from multiple session, provided object is present in the second level cache.

The second level cache is maintained at the SessionFactory level, which is used to open sessions, hence every session is

linked to SessionFactory. This cache is opposite to first level cache which is by default enabled in Hibernate, this one is by

default disabled and you need to configure the second level cache in Hibernate configuration file to enable it.

The second level cache is provided with the help of caching providers e.g. EhCache and OSCache. If you look at the cache

package in Hibernate, you can see the implementation of Caching related interfaces by these providers. Depending upon which

cache you want to use, you can configure them in the Hibernate Configuration file.

Once configured, every request for an object will go to the second level cache if it is not found in the first level cache.

It won't hit the database without consulting second level cache, which means improved performance.

It's very important for a Java and Hibernate developer to know about Caching in Hibernate. It's not just important from

Interview point of view but also from the application development and performance improvement point of view. You will often face

performance related challenges in a real world application which contain millions of records, by correctly configuring Hibernate

sessions and writing code which make use of caching, your Java application can float above water even in the case of a significant

load. If you want to learn more about Hibernate performance, I suggest reading I suggest reading High-Performance Java Persistence by

Vlad Mihalcea, one of the best and up-to-date resources on hibernate performance at the moment.

=========================================================================================================================================

What is Second-level Cache in Hibernate?

This is one of the first interview question related to caching in Hibernate, you can expect few more. Second-level

Cache is maintained at SessionFactory level and can improve performance by saving few database round trip. Another worth

noting point is that second level cache is available to the whole application rather than any particular session.

What is query cache in Hibernate?

This question Sometimes asked as a follow-up of last Hibernate Interview question, QueryCache actually stores the

result of SQL query for future calls. Query cache can be used along with second level cache for improved performance.

Hibernate support various open source caching solution to implement Query cache e.g. EhCache.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qoCPgWU

=========================================================================================================================================

Can we make a Hibernate Entity Class final?

Yes, you can make a Hibernate Entity class final, but that's not a good practice. Since Hibernate uses a proxy pattern for performance

improvement in the case of the lazy association, by making an entity final, Hibernate will no longer be able to use a proxy, because

Java doesn't allow extension of the final class, thus limiting your performance improvement options. Though, you can avoid this penalty

if your persistent class is an implementation of an interface, which declares all public methods defined in the Entity class.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qoJbrbw

=========================================================================================================================================

Why it's important to provide no argument constructor in Hibernate Entities?

Every Hibernate Entity class must contain a no argument constructor,

because Hibernate framework creates an instance of them using Reflection API, by calling Class.newInstance() method.

This method will throw InstantiationException if it doesn't found any argument constructor inside Entity class.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz54qoXxXN0

=========================================================================================================================================

=========================================================================================================================================

=========================================================================================================================================

=========================================================================================================================================

what is hibernate proxy??=========================================================================================================================================

By default Hibernate creates a proxy for each of the class you map in mapping file. This class contain the code to invoke JDBC. This class is created by hibernate using CGLIB.

Proxies are created dynamically by subclassing your

object at runtime. The subclass has all the methods of the parent, and when

any of the methods are accessed, the proxy loads up the real object from the

DB and calls the method for you. Very nice in simple cases with no object

hierarchy. Typecasting and instanceof

work perfectly on the proxy in this case since it is a direct subclass.

//---------------------------

Spring boot remote web service call ---

33. Calling REST services

If you need to call remote REST services from your application, you can use Spring Framework’s RestTemplate class. Since RestTemplate instances often need to

be customized before being used, Spring Boot does not provide any single auto-configured RestTemplate bean.

It does, however, auto-configure a RestTemplateBuilder which can be used to create RestTemplate instances when needed.

The auto-configured RestTemplateBuilder will ensure that sensible HttpMessageConverters are applied to RestTemplate instances.

Here’s a typical example:

@Service

public class MyBean {

private final RestTemplate restTemplate;

public MyBean(RestTemplateBuilder restTemplateBuilder) {

this.restTemplate = restTemplateBuilder.build();

}

public Details someRestCall(String name) {

return this.restTemplate.getForObject("/{name}/details", Details.class, name);

}

}

[Tip]

RestTemplateBuilder includes a number of useful methods that can be used to quickly configure a RestTemplate. For example, to add BASIC auth

support you can use builder.basicAuthorization("user", "password").build().

33.1 RestTemplate customization

There are three main approaches to RestTemplate customization, depending on how broadly you want the customizations to apply.

To make the scope of any customizations as narrow as possible, inject the auto-configured RestTemplateBuilder and then call its methods as required. Each method call returns a new RestTemplateBuilder instance so the customizations will only affect this use of the builder.

To make an application-wide, additive customization a RestTemplateCustomizer bean can be used. All such beans are automatically

registered with the auto-configured RestTemplateBuilder and will be applied to any templates that are built with it.

Here’s an example of a customizer that configures the use of a proxy for all hosts except 192.168.0.5:

static class ProxyCustomizer implements RestTemplateCustomizer {

@Override

public void customize(RestTemplate restTemplate) {

HttpHost proxy = new HttpHost("proxy.example.com");

HttpClient httpClient = HttpClientBuilder.create()

.setRoutePlanner(new DefaultProxyRoutePlanner(proxy) {

@Override

public HttpHost determineProxy(HttpHost target,

HttpRequest request, HttpContext context)

throws HttpException {

if (target.getHostName().equals("192.168.0.5")) {

return null;

}

return super.determineProxy(target, request, context);

}

}).build();

restTemplate.setRequestFactory(

new HttpComponentsClientHttpRequestFactory(httpClient));

}

}

Lastly, the most extreme (and rarely used) option is to create your own RestTemplateBuilder bean. This will switch off the auto-configuration of a RestTemplateBuilder and will prevent any RestTemplateCustomizer beans from being used.

//-------------------------------------------------------------------------------------------------------------------------------

What is the purpose of serialization?

Answer: Serialization is the conversion of an object to a series of bytes

, so that the object can be easily saved to persistent storage or streamed across a communication

link. The byte stream can then be deserialised – converted into a replica of the original object.

Source | Example

What is the difference between JDK and JRE?

Answer: Java Development Kit (JDK) is the most widely used Java Software Development Kit. Java Runtime Environment (JRE) is an

implementation of the Java Virtual Machine which executes Java programs.

Source | JDK Wiki | JVM Wiki

What is the difference between equals() and “==” ?

Answer: Equals is intended to check logical equality and == checks if both references point to same object. (Thanks Sandeep)

a == b; // Compares references, not values.

a.equals(b); // Compares values for equality.

Source

When will you use Comparator and Comparable interfaces?

Answer: java.util.Comparator and java.lang.Comparable

java.util.Comparator compares some other class’s instances,

while java.lang.Comparable compares itself with another object.

Source | Example

What is the wait/notify mechanism?

Answer: This deals with concurrent programming. The wait() and notify() methods are designed to provide a mechanism to allow a thread to be block until a specific condition is met.

However, java.util.concurrent should be used instead of wait() and notify() to reduce complexity.

Source | Java API | Java Technical Article

What is the difference between checked and unchecked exceptions?

Answer:

In general, unchecked exceptions represent defects in the program (bugs), which are normally Runtime exceptions.

Furthermore, checked exceptions represent invalid conditions in areas outside the immediate control of the program.

Source

What is the difference between final, finally and finalize?

Answer: “final” is the keyword to declare a constant AND prevents a class from producing subclasses. (Thanks Tom Ellis)

“finally” is a block of code that always executes when the try block is finished, unless System.exit() was called. finalize() is an method that is invoked before an object is discarded by the garbage collector.

Source | Final Usage |Finally Usage | Finalize()

What is the difference between web server and app server?

Answer: A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.

Source

Explain the Struts1/Struts2/MVC application architecture?

Answer: Struts was adopted by the Java developer community as a default web framework for developing web applications

The MVC(Model–view–controller) an application that consist of three distinct parts. The problem domain is represented by the Model. The output to the

user is represented by the View. And, the input from the user is represented by Controller.

Source

What is the difference between forward and sendredirect?

Answer: Both method calls redirect you to new resource/page/servlet. The difference between the two is that sendRedirect always sends a header back to the client/browser, containing the data in which you wanted to be redirected.

Source

How does a 3 tier application differ from a 2 tier one?

Answer: Tiers are the physical units of separation or deployment, while layers are the logical units of separation.

Imagine that you’re designing an e-commerce website. A 3 tier architecture would consist of web pages, a web server and a database, with the corresponding 3 layers being the “Presentation”, “Business Logic” and “Database” layers.

If you take the database tier and layer out then your have a 2 tier architecture.

Source

How does the version control process works?

Answer: Initiate, pull, branch, merge, commit, push.

(Init) Make your own repository. (Pull) Download an existing repository from a url. (Branch / Merge )Make revisions. Commit then push your modifications.

Git Cheat Sheet

What is the difference between JAR and WAR files?

Answer: JAR files (Java ARchive) allows aggregating many files into one, it is usually used to hold Java classes in a library.

WAR files (Web Application aRchive) stores XML, java classes, and JavaServer pages for Web Application purposes.

Source

What is a Left outer join?

Answer: This deals with SQL. Left outer join preserves the unmatched rows from the first (left) table, joining them with a NULL row in the shape of the second (right) table.

Source | Joins Wiki

What is the difference between UNION and UNION ALL?

Answer: This deals with SQL. UNION only selects distinct values, UNION ALL selects all values.

Source | Example

http://pulseabledev.mckinleyequipment.com