```
@DynamicUpate(value=true)
It is used to generate the update query only for the fields on which the setter
injection is
performed.

Entity object fields are
  sid,sname,sage,saddress

The entity object setter method used is main method is
      student.setSaddress("RCB");

Query generated is
update
        Student
    set
        saddress=?
    where
        sid=?

Note: Within the transaction, persitence object would be synchronized to a row in
the database.

Alternative ways available to configure ORM
-------------------------------------------
a. using Programmatic approach(Pure java approach)
b. using Properties file(application.properties)
c. using XML approach(hibernate.cfg.xml)

1) Programmatic Approach
In Programmatic approach, to provide configuration details, first we have to create
Configuration class object then we
have to set all hibernate properties to Configuration class object explicitly by
using the following method.
                  public void setProperty(String prop_Name, String prop_Val)
To add mapping file name and location to Configuration file we have to use the
following method.
                  public void addResource(String mapping_File_Name)
Note: If we are using annotations in place of mapping file then we have to use the
following method to add annotated class.
                  public void addAnnotatedClass(Class class)

In Programatic approach, if we want to change configuration details like connection
properties or database properties,....
then we have to perform modifications in JAVA code, if we perform modifications in
JAVA code then we must
recompile the java application , it is not suggestible in enterprise applications.
            To overcome the problem we have to use Declarative approach.

2) Declarative approach:
In Declarative approach, we will provide all configuration details in either
properties file or in XML file then we will send
configuration details to Hibernate application.
There are two ways to provide configuration details to Hibernate Applications in
Declarative Approach.
            1) By using properties file.
            2) By Using XML file.

1) Using properties file in Declarative Approach:
In This approach, to provide all hibernate configuration details , we have to
```

declare a properties file with the default name
"hibernate.properties" under "src" folder . In this context, when we create Configuration class object ,
Hibernate Software will search for the properties file with the name
"hibernate.properties" and get all the configuration
details into Configuration class object.
In this approach we must add mapping file explicitly to Configuration File by using the following method.

```
            public void addResource(String mapping_File-Name)
```

Note:
If we change name and location of properties file from hibernate.properties to some other "abc.properties" then we have
to give that intimation to Hibernate software, for this, we have to create FileInputStream and Properties class object
then we have to set Properties object to Configuration class object by using the following method.

```
                        public void setProperties(Properties p)
```

Note:
In hibernate applications, if we use "properties" file then we are able to provide only hibernate connection propeties,
dialect propeties,... , but, we are unable to provide mapping properties,... through properties file, to provide mapping properties
we have to use java methods like addResource(--) or addAnnotatedClass(--) methods from Configuration class.

In Hibernate applications, if we want to provide all configuration details in declarative manner then we have to use XML file approach.

If we want to use xml file to provide all configuration details then we have to use configure() method to get all configuration
details from xml file. Here configure() method will search for xml file with the default name hibernate.cfg.xml under src folder inorder to get configuration details.

If we change the default name and location of hibernate configuration file then we have to pass that name and location to configure(--) method as parameter.

Note:
if we place all the above approach(hibernate.cfg.xml,hibernate.properties,programmatic approach) then which one will be applied to set up the jdbc environment by Hibernate?
=> cfg.configure() is called in the code after setting the properties then hibernate.cfg.xml file will be used,otherwise programming
   logic(cfg.setProperty()) will be used.

Primary Key Generation Algorithms in Hibernate
==
Primary key is single column or Collection of columns in a table to recognize the records individually.
In Database applications, to perform the database operations like retriving a record, updating a record, deleting a record,....
we need primary key and its value.
In Database applications, we are unable to give option to the users to enter primary key values , because there is no
guarantee for the data entered by the users whether it is unique data or not, but, in Database tables only

unique values are accepted by primary key columns.

To give guarantee for uniqueness in primary key values we have to use Primary key generation algorithms.

Almost all the Persistence mechanisms like Hibernate, Ibatis, Open JPA, Toplink,.... are having their own implementation for
primary key generation algorithms.

Hibernate has provided support for the following primary key generation algorithms inorder to generate primary key values.
1. assigned(default)
2. increment
3. sequence
4. identity
5. hilo(removed from HB5.X)
6. native.
7. seq-hilo
8. select
9. UUID
10.GUID
11. foreign

Hibernate has represented all the above primary key generation algorithms in the form of a set of predefined classes provided
      in "org.hiberante.id" package.
If we want to use any of the above algorithms in Hibernate applications then we have to configure that algorithms in
      hibernate mapping file by using the following tags.

There are 3 types of generators in hibernate
1. Hibernate supplied generators
      a. @GeneratedValue
      b. @GenericGenerator
2. JPA supplied generators
      a. @GeneratedValue
3. Custom generator(** realtime projects)

1. assigned
This algorithm is default primary key generation algorithm in hibernate applications, it will not required configurations in
mapping file.
This algorithm will not have its own mechanism to generate primary key value , it will request to Client Application to
provide primary key value explicitly.
This algorithm is able to support for any type of primary key values like short, int, long, String,.....
This algorithm is supported by almost all the databases like Oracle, MySQL, DB2, ...
This algorithm is not required any input parameter.
This algorithm is represented by Hibernate in the form of a predefined class "org.hibernate.id.Assigned".

```
@Id
@GenericGenerator(name = "gen1", strategy = "assigned")
@GeneratedValue(generator = "gen1")
private Integer eid;
```

2.
 increment

This primary key generation algorithm is able to generate primary key value by incrementing max value of the primary key column.

New_Val = max(PK_Column)+1

This alg is able to generate primary key values of the data types like short, int, long,...
This alg is not required any input parameter to generate primary key values.
This alg is supported by almost all the databases which are supporting numeric values as Primary key values.
This alg is representned by Hibernate Software in the form of a short name "increment" and in the form of a predefined class like
"org.hibernate.id.IncrementGenerator".

```
@Id
@GenericGenerator(name = "gen1",strategy = "increment")  // To specify details HB Specific generator
@GeneratedValue(generator = "gen1") // To apply generators on @Id field
private Integer eId;
```

3. sequence
This primary key generation algorithm is able to generate primary key value on the basis of the sequence provided by the underlying Database.
This alg is able to generate primary key values of the data types like short, int, long,...
This alg required "sequence" input parameter with the sequence name as value inorder to generate primary key value.
If we have not provided any sequence name as input parameter then this alg is able to take "hibernate_sequence" as default sequence name,
here developers must create "hibernate_sequence" in database explicitly.
This alg is supported by almost all the databases which are supporting sequences like Oracle, DB2,....
This alg is represented by Hibernate Software in the form of a short name like "sequence" and in the form of a predefined class like
"org.hibernate.id.SequenceGenerator".

1>
```
@Id
@GenericGenerator(name = "gen1", strategy = "sequence",
                              parameters ={
                                                  @Parameter(value =
"eid_seq",name="sequence_name")
                                               }
                              )
@GeneratedValue(generator = "gen1")
private Integer eid;
```

2>
```
@Id
@GenericGenerator(name = "gen1", strategy = "sequence")
@GeneratedValue(generator = "gen1")
private Integer eid;
```

4.
identity
This alg is able to generate primary key values on the basis on the underlying database table provided identity column.

Note:

Identity column is a primary key Column with "auto_increment" capability.
This alg is able to provide the primary key values of the data types like short, int, long,....
This alg is not required any input parameter.
This alg is supported by almost all the databases which are supporting Identity column.
EX: MySQL.
To represent this alg, Hibernate has provided "identity" as short name and "org.hibernate.id.IdentityGenerator" as predefined class.

```
@Id
@GenericGenerator(name = "gen1",strategy = "identity")  // To specify details HB Specific generator
@GeneratedValue(generator = "gen1") // To apply generators on @Id field
private Long policyId;
```