list() or getResultList()

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

1. It internally uses eager loading for bulk operations.

2. It returns the collection directly.

3. Generates only one query to get all the records.

4. suitable for good performance.

5. It won't generate proxy object.

6. It is not deprecated and it is the industry standard approach.

iterate()

=======

1. It internally uses lazy loading for bulk operation

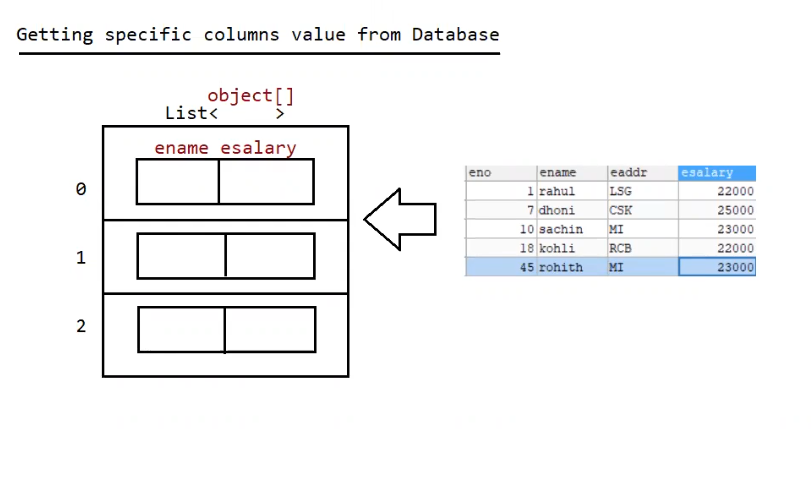
2. It returns the iterator pointing to collection object.

3. Generates n+1 query to get all the records.

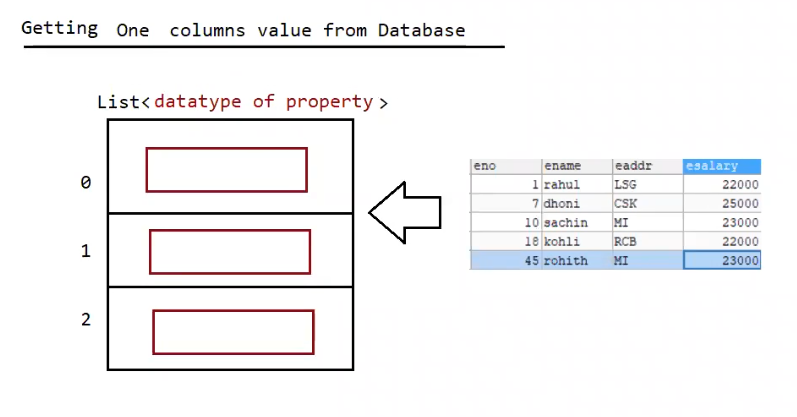
4. Not suitable, it degrades the performance.

5. It generates Proxy object.

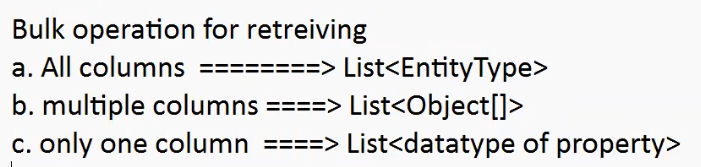
6. It is deprecated because of performance issue.



Eg: HibernateBulkOperationSpecificColumn



Eg: HibernateBulkOperationSingleColumn



**When to Use uniqueResultOptional()**

* Use it when your query **must return either zero or one result** (e.g., searching by primary key or unique column).

Note: To avoid null checking in our application,we can use JDK8 supplied api called "Optional".

Eg: HibernateBulkOperationWithOptional

Note: we should use get()/load() if we are getting record based on primary key value,

we can use uniqueResult()/uniqueResultOptional() for **any query**, including those **based on primary key**, if you're using HQL or Criteria API instead of get().

Eg: HibernateBulkOperationUpdate

Eg: HibernateBulkOperationDelete

HQLInsert operation

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

It is not possible to insert one record to the database directly using insert query, becoz linking generators with HQL insert query is not possible. so we use session.save() method to insert a record.

We can use HQL insert query to insert bulk record into one db table by selecting them from another db table.

eg: insert into .... values (query is not given)

insert into ... SELECT FROM ...... (given to perform bulk operation)

Eg: HibernateBulkOperationInsert

NamedHQLQuery

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

=> So far our HQL query is specific to one session object becoz query object is created having hard coded HQI query on session object.

=> To make our HQL query accessible and executable through multiple session objects of Mulitple DAO classes or client apps we need to go for "NamedHQL" query.

=> We defined NamedHQLQuery in mapping file using <query> tag or in Entity class using "@NamedQuery" havinig logical name and we access and execute that HQL query in DAO class.

Eg: HibernateNamedHqlQuery

NativeSQL Query

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

=> It is given to execute plain SQL queries that are supported by underlying DB S/w.

=> We need to use these operations only when it is not possible through HQL

eg: inserting a single record.

=> It supports both select and non-select operation

=> These queries performance is bit good compared to HQL because they go to sql directly without any conversion.

=> We write a query using table name and column names.

Eg: HibernateNamedNativeSqlQuery

Eg: HibernateNamedNativeSqlQuerySelect2

Eg: HibernateNamedNativeSqlQueryDelete

Criterion Api

==========

SRO ==== > hibernate persistence methods ( single row operation )

bulkoperations => we use HQL(query written using classname and properties) to write queries.

In case of Criterion api, we can perform both singlerow and bulkoperations without using any queries just like java statements.

=> Criterion api will generate SQL queries based on the given entity classnames and properties name

It does-not support non-select operation, it supports only select operation.

Using Criteria object we can add 3 object

a. Criterion objects(for where clause condition)

b. Project objects(for scalar select operation)

c. Order object( for orderBy operations)

There are 2 modes of writing Criterion api

a. HB QBC(Query by Criteria) === > specific to hibernate only

b. JPA QBC === > common to all ORM framework

Eg: HibernateCriterianAPI

Eg: HibernateCriterianAPISelectOperationWithCondition

Pagination

=========

Displaying large volume of records into muliptle pages is called as "pagination".

Hibernate supports pagination through QBC

1. setFirstResult(int pageNo)

2. setMaxResult(int maxNo)

Eg: HibernateCriterianAPIPageNation

Eg:JPAQueryByCriteriaForAllColumns

Eg: JPAQueryByCriteriaForSpecificColumns

StoredProcedure:

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

Eg: HibernateStoredProcedureAllColumns

Eg: HibernateStoredProcedureSpecificColumns

Locking in hibernate

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

If multiple apps or threads simultaneoulsy accessing and manipulating the records there is a possibility of getting concurrency problem.

To Avoid this problem we need to use "Locking " of a record in hibernate.

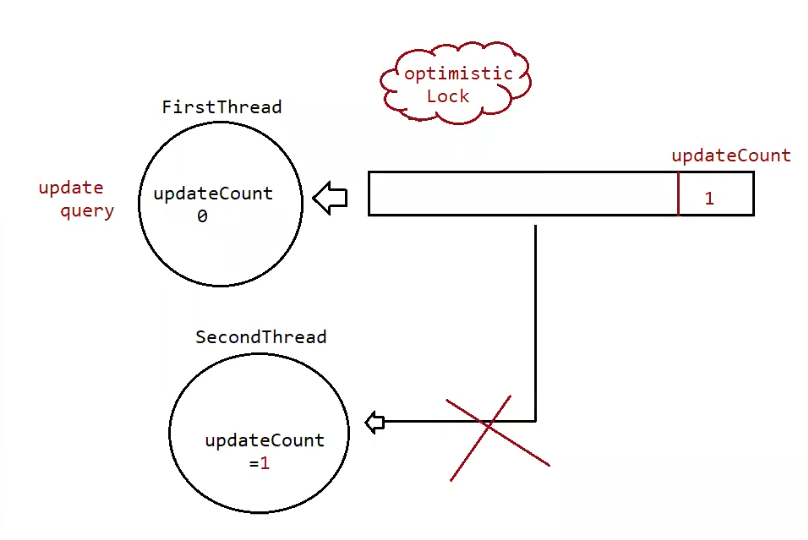
Hibernate supports 2 levels of Locking

a. Optimistic Locking

=> It allows secondthread simultaneously to access and modify the record, then first thread notices the modification and throws "Exception".

=> To enable this locking we need to use "@Version" in our application.

=> if we use @Version then automatically optimisitic locking will be achieved.

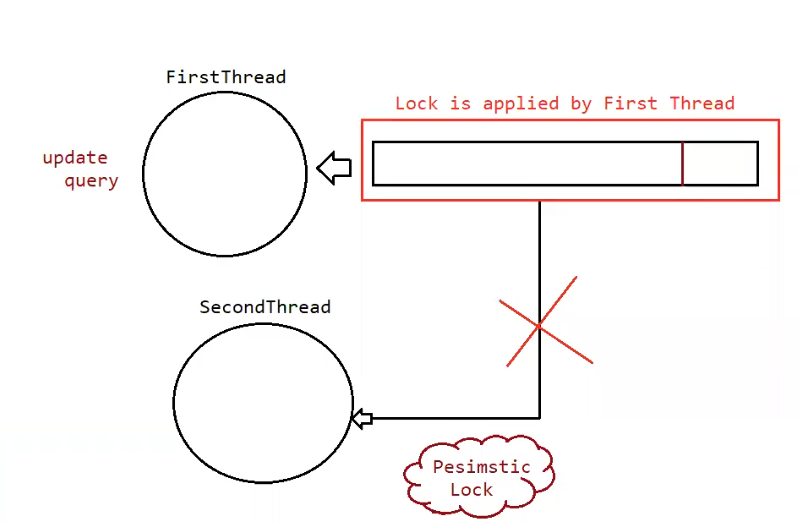


Eg: HibernateOptimisticLocking

b. Pesimistic Locking

=> It will allow FirstThread to Lock the record itself,so if the second thread tries to access and modify the record then it should throw "Exception".

=> To enable this locking we need to use session.get(Class c,Serializable s,LOCKMODE.UPGRADE\_NOWAIT) as the third argument value



Eg: HibernatePesimisticLocking

Mapping in hibernate

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

In realtime applications, we use mapping to link two classes and these linking at the database side will happen in the form of primary-key foreign-key relationship.

At the java side, we can link 2 classes through "Association".

At the database side we don't have these linking,but we have something called as "Primary-Foreing" key relationship.

To support this feature at the hibernate side we have "Mapping".

There are 4 types of hibernate mapping

a. One-One Association Mapping

b. One-Many Association Mapping

c. Many-One Association Mapping

d. Many-Many Association Mapping

One-Many Association Mapping

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

It refers to relationship b/w 2 entities where one instance of one entity should be mapped with exactly one instance of another entity.

eg: One Employee has One Account

Annotation used is :: @OneToOne(cascade = CascadeType.ALL)

cascade specifies, if we delete employee table automatically account table also should be deleted

Eg: HibernateOneToOneMapping

Eg: HibernateOneTOOneMappingSelect

One-Many Association Mapping

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

It refers to relationship b/w 2 entities where one instance of one entity should be mapped with multiple instances of another entity.

eg: Single Department has Multiple Employees

Annotation used is :: @OneToMany(cascade = CascadeType.ALL)

Eg: HibernateOneToManyMapping

Eg: HibernateOneToManyMappingSelect

Many-One Association Mapping

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

It refers to relationship b/w 2 entities where mulitple instance of an entity should be mapped with exactly one instance of another entity.

eg: Multiple Students have joined with Single Branch

Annotation used is :: @ManytoOne(cascade = CascadeType.ALL)

Eg: HibernateManyToOneMapping

Eg: HibernateManyToOneMappingSelect

Many-Many Association Mapping

-------------------------------------------

It refers to relationship b/w 2 entities where mulitple instances of an entity should be mapped with multiple instances of another entity.

eg: Multiple Students have joined with Mulitple Courses.

Annotation used is :: @ManytoMany(cascade = CascadeType.ALL)

Eg: HibernateManyToManyMapping

Eg: HibernateManyToManyMappingSelect