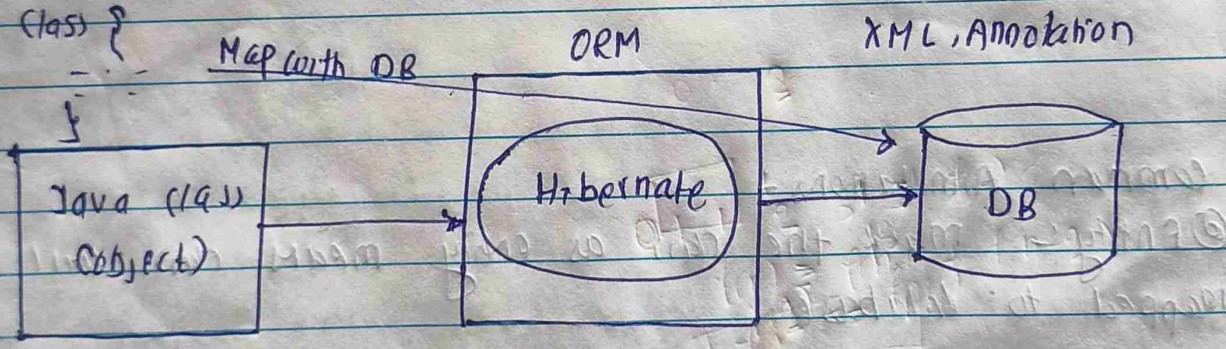
Gavin King in 2001





# ⊛ Hibernate Framework ⊛

- Hibernate is a java framework that simplifies the development of java app. to interact with DB.
- Open source, lightweight, ORM (Object Relational mapping) tool, implement JPA (Java persistence API)
- Hibernate is non-invasive framework, means it wont forces the programmer to extend/implement any class/interface.
- can load any app.
- JDBC - Develope has to depends on DB (what is name of DB, table name, column name etc)
- Hibernate that's why comes into picture.
- Hibernate use internally jdbc.




To work on Hibernate, add depedency
- Hibernate core Relocation
- mysql connector.

Hibernate configuration file xml based :-
- create hibernate.cfg.xml file in src/main/java.
- Download dtd file of hibernate, copy <!DOCTYPE> tag, paste on xml file.
- Give basic configuration in xml file.

If we create only file name with hibernate.xml then
- configure("hibernate.xml")

Java based configuration :-

④ Comman Annotations :-

@Entity :- Mark the class as entity means class will be mapped to database.

@Id :- primary key of Entity

@GeneratedValue :- Define strategy for generating primary key.
GenerationType.IDENTITY strategy means database responsible for generating primary key.

@Table :- define table and other properties for an entity. customization of table name, schema other options.

@Column :- define column specific properties, customization of column name length, nullable, unique, comments.

Eg.
@Column (name = "", nullable = false, unique = true)
private int money;

⑤ @Transient :- entity attribute would not take part in database (not persist).

⑥ @Temporal :- specify the type of temporal data to be stored in a date or time attribute.

⑦ @Lob :- map large object fields in entities. (e.g. BLOB (Binary large object) or CLOB (character large object)

Fetch data from databases :-

get() :- if data not present gives null.
It loads data on session, means if we r not using data, still retrive the values.

load() :- if data not present, gives objectNotFoundException.
gives proxy object means, if we are not using data, it will retrive the values.

CRUD operation :-

insert :- session.save() or session.persist().
read :- session.createQuery("from Employee");
update :- update() or merge() or save()
delete :- delete()

Relationship / mapping :-

Relationship betn two tables (primary key & foreign key).

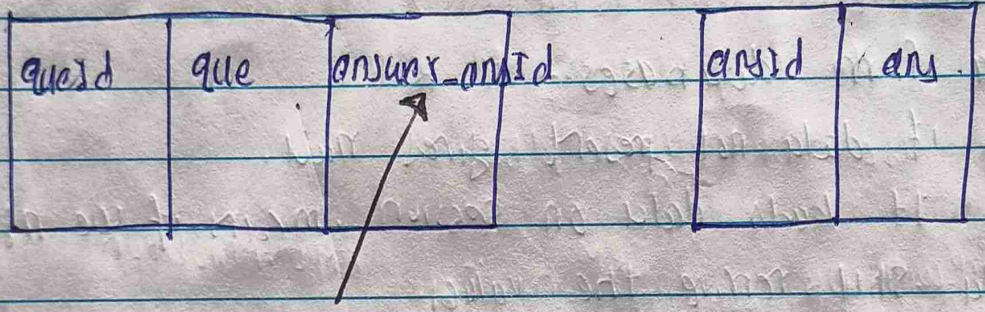
## Relationship / Mapping :-

### @ one to one :-

```
class Question {
    int quesId;
    String que;
    @oneToOne
    private Answer answer;
}
```

```
class Answer {
    int ansId;
    String ans;
}
```

This will create




This name derived from answer_ansId (private Answer answer & int ansId) which is equal to ansId

Internally @oneToOne uses @JoinColumn to give implicit name.

We can change column name by @JoinColumn (name="ansId"). Now I can easily recognize related table & their mapping.

@ oneToOne (cascade = CascadeType.ALL)

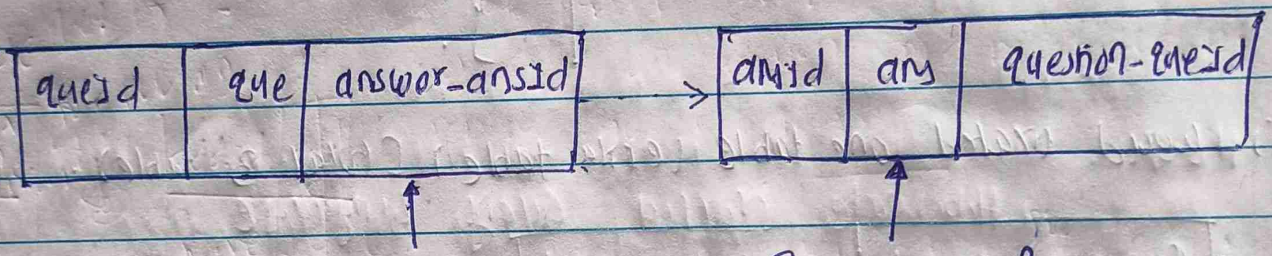
- If I would have declared this annotation in Question class, I didn't need to save Answer object.
- Which means that if we perform any operation on one table & then specific changes related to related table also changes.

Now this is become unidirectional. means Question table changes Reflect on Answer, vica-versa not.

Now

```
class Answer {
int ansId;
String ans;
@OneToOne
Private Question question;
```

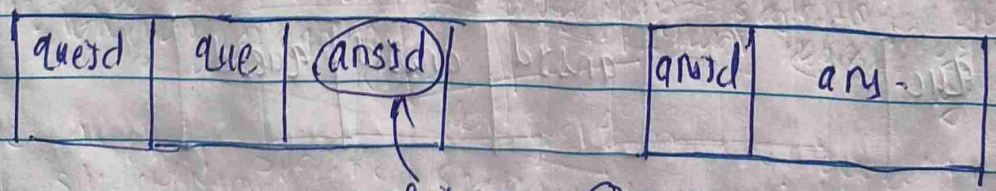
Now in this case, two table will created




Here @OneToone of Question & @OneToOne of Answer

- This is not good practice (approach) leads to data consistency. (breaks privacy).
- So I will use @OneToone(mappedBy = "answer") in Answer class. this means that the whoever referring me this will take responsibility (this is not my responsibility).

Now overall table is




if I use @JoinColumn("ansId").

Now this is become bidirectional.

Note :- Don't use @JoinColumn / @JoinTable & mappedBy together → unexpected behaviour / data inconsistency.

@OneToMany :

```
class Question {
    ...
    @OneToMany
    private List<Answer> answers;
}
```

```
class Answer {
    ...
}
```

This would make one table extra table (total 3 table).

| Question_quesId | answers_ansId |
|---|---|

Now I need to customize table.

@JoinTable(name = "customizedTable", joinColumns = @JoinColumn(name = "quesId"), inverseJoinColumns = @JoinColumn(name = "ansId"))

If you don't want separate table then

| quesId | que |
|---|---|

Question class

| ansId | ansId | quesId |
|---|---|---|

Answer class

mappedBy = "question"

↳ Referenced class will make one column again (quesId) @JoinColumn("quesId").

Note: Always tries to write sequence code (Question first, Answer then). (null id problem → assign constructor, getter-setters properly)

(a) One to one → create extra one column.
(b) One to many → create one extra table
(c) Many to Many → create like
(d) Many to one → create extra one column.

(*) fetch type:
FetchType.LAZY: It fetches the child entities lazily, that is at the time of fetching parent entity it just fetches proxy (created by cglib or any other utility) of the child entities & when you access on property of child entity then it is actually fetched by hibernate.
- improves performance by avoiding unnecessary computation & reduce memory requirement.

FetchType.EAGER:
- It fetches the child entities along with parent.
- Depends on situation.

(*) HQL: Hibernate Query Language
- Similar to SQL but instead on tables & columns, works with persistent objects & their properties.
- HQL queries are translated by hibernate into conventional SQL queries, which in turn perform action on databases.
- We can write complicated (conditional) queries.

Query ← class name.

St = Session.createQuery("from Student", Student.class); ← Gives null if not found.

St.getResultList(); ← Gives list of object.

St.uniqueResult(); ← gives single object.

for single attribute,

Query<String> stQuery = Session.createQuery("select city from student where name = 'viraj'", String.class);

↑ No. of type (data) of city.

⊛ HCQL :- Hibernate criteria Query language

- In HQL still we were using SQL queries.

HCQL is alternative to HQL & allows us to build & execute queries in a type-safe & object-oriented manner.

- createCriteria() method is used to create.

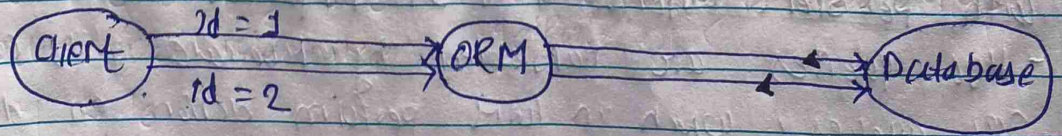
↳ Now, it is deprecated.

- CriteriaBuilder can we used instead of that.

⊛ Caching :-

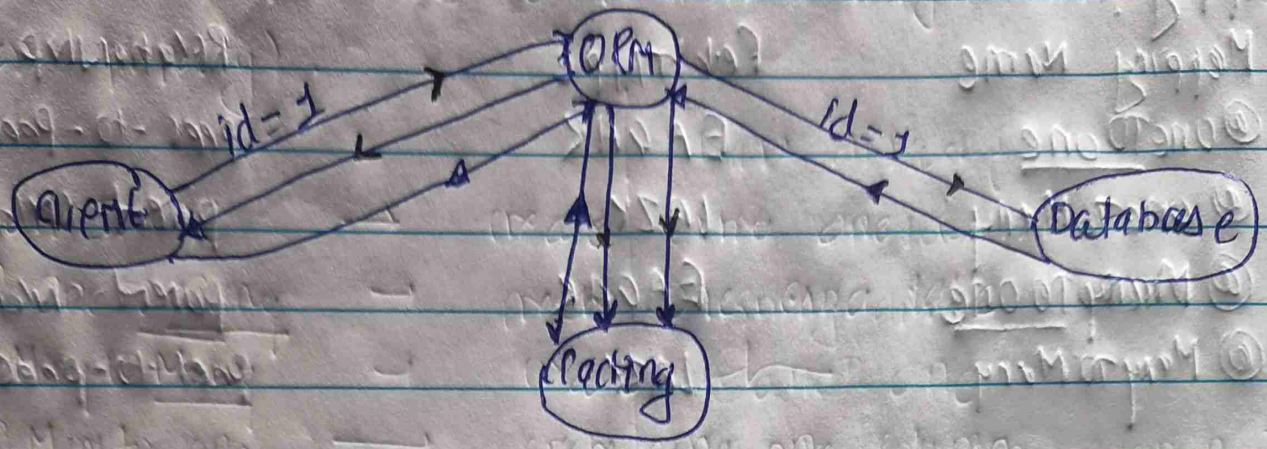
It is useful when we have to fetch the same data multiple times.

without caching:




Here for same data, we are sending two different request

With Caching :-




With cache ORM will check first data at caching if not present, it will to DB.

Types :-
First level cache :-
Second level cache
Query level cache.

First level cache :-
- Session object holds the data (first level cache data), enabled default, not application for entire application.
- An application can use many session object.

Second level cache :-
SessionFactory object holds this data, for entire application. but need to define explicitely.

Query level cache :-
For query results that integrates closely with the second level cache.

EH (Earl Hibernate) cache, Swarm cache, OS cache, JBoss cache.

# Hibernate with spring boot

Mapping :-

| Mapping Name | Fetch Type | Relationship |
|---|---|---|
| @oneToOne | EAGAR | peer-to-peer |
| @oneToMany | LAZY | parent-child |
| @ManyToOne | EAGAR | parent-child |
| @ManyToMany | LAZY | peer-to-peer |

- Some times we cannot make @oneTo one LAZy directly. I got error here (usually if there are more than 1 mapping in single entity.

What I did

optional = "false" :- when optional is set to false, it means that the relationship is not optional, there must always be a corresponding entity on the other side of the relationship. If the associated entity doesn't exist it would result in an exception (entityNotFoundException).

@MapsId

primary key of dependent entity (e.g. AuthorInfo) should be mapped with to primary key of the parent Entity (e.g. Author) (Author & AuthorInfo will have same primary key column)

@JoinColumn (name = " ")

Customize the Foreign key column

mappedBy :-

Define the owning side of a bidirectional relationship.

In bidirectional relationship

- owning side
- inverse side

The mappedBy attribute used on the owning side of the relationship to indicate that relationship is being mapped by corresponding attribute on inverse side

Error I got

LazyInitializationException

trying to access a lazily loaded property of an entity outside of the original Hibernate session scope

E.g.

Book → Author → AuthorInfo (lazy loaded)

Use

@Transactional :

Mark methods or classes where transaction should be applied. Spring will ensure that transaction is started before the method & committed (or rolled back) after the method completes.

StackOverflowError :

If we trying to print object (eg. TransactionDetails) & this object has Fetch loading type, so it fetch all info.

Note :

- Delete works with hibernate, if we try directly delete data through database, it might not be work.
- In bidirectional mapping we have to save both entity object through their respective repository object, otherwise mapping will not happen.
- Think about sequence properly while saving object (when create which object, assign, getter setter, cascadeAll).
- Try to create (foreign) table on owning side in @ManyToMany relationship, it is better approach.