infogain

ORM With Hibernate

Objectives

- At the end of the course participants will be able to
  - Define cascade attribute
  - Identify relationships using Association Mappings
  - HQL
  - Hibernate Second Level cache

Cascade Attribute

- **Cascade Attribute**
  - It is mandatory to getting the relation between parent and child class objects.
  - when ever we apply relationship between objects, cascade attribute transfers operations done on one object onto its related child objects.
  - if we write **cascade = "all"** then all operations like insert, delete, update at parent object will be effected to child object also.
  - Default value of **cascade ="none"** means no operations will be transfers to the child class

Association Mapping

- Mapping put relationship between two entities objects of two pojo classes
- The mappings are applied to express the various different ways of forming associations in the underlying tables.
- The advantage of putting relation ship between objects is, we can do operation on one object, and the same operation can transfer onto the other object in the database

Hibernate Association

- **Has-A Association**
  - One to one
  - One to Many
  - Many to one
  - Many to many
- Association can be unidirectional or bidirectional.
- **Unidirectional:**
  - In case of unidirectional association one entity is associated with another. So operation can be performed in one way.

Association (contd..)

- **Unidirectional Association**
  - public class Employee{
    private String name;
    private String phone;
    private Department dept;
    //getters and setters
    }
  - public class Department{
    private String deptId;
    private String deptName;
    //getters and setters
    }
- Here Department object is declared as property of the employee class. Every employee is mapped to Department but Department doesn't know about Employee, because of Department doesn't have any association with Employee(i.e.,Department has no Employee). then it unidirectional.

Association (contd..)

- **Bidirectional Association**
  - public class Employee{
    private String name;
    private String phone;
    private Department dept;
    //getters and setters
    }
  - public class Department{
    private String deptId;
    private String deptName;
    private Employee emp
    //getters and setters
    }
  - Here Employee class have Department and Department can have Employee so that every Employee has Department and each Department is mapped to Employee so it is bidirectional.

Mapping Elements

- Hibernate provides following annotations for the mapping of the Has-A relations.
- **@Onetoone**
  - is used to specify onetoone relation between the entity class and the annotated data member.
- **@Onetomany**
  - is used to specify the onetomany relation between the entity class and the annotated data member.
- **@Manytoone**
  - is used to specify manytoone relation between the entity class and the annotated data member.
- **@manytomany**
  - is used to specify manytomany relation between the entity class and the annotated data member.

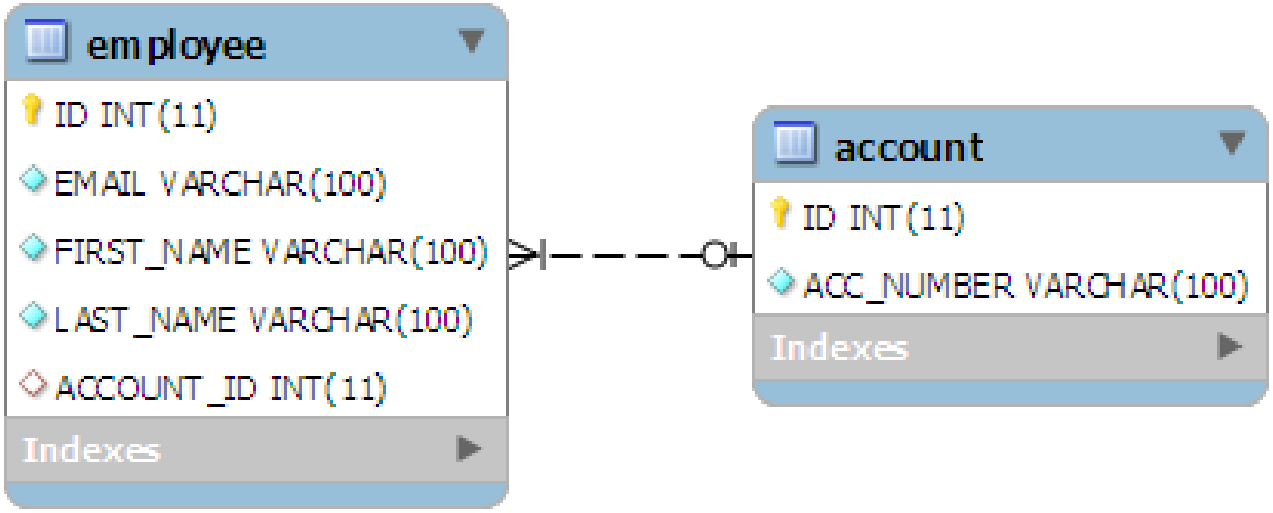
One-To-One Mapping

- **Can be implemented in several ways**
  - Using foreign key
  - Using a common join table
  - Using shared primary key

- **Using foreign key association**
  - Used to link two tables through primary key foreign key relation.
  - In this association, a foreign key column is created in owner entity.
  - The join column is declared with the @JoinColumn annotation
  - If no @JoinColumn is declared on the owner side, the defaults apply.

One-To-One Mapping (contd..)

- For example,
- if we make Employee entity owner, then a extra column “ACCOUNT_ID” will be created in Employee table. This column will store the foreign key for Account table



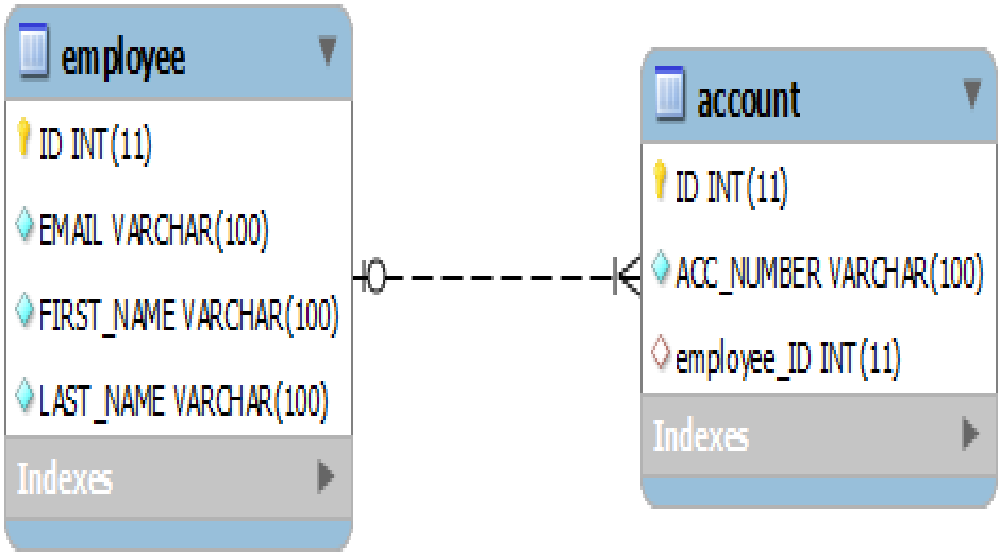
- To make such association, refer the account entity in EmployeeEntity class as follow:
  - @OneToOne
  - @JoinColumn(name="ACCOUNT_ID")
  - private Account account;

One to Many Association

- First entity can have relation with multiple second entity instances but second can be associated with only one instance of first entity.
- OneToMany unidirectional relation between objects can be mapped using primary key foreign key mapping
- In OneToMany unidirectional relation ,owner entity contains a collection of owned entities as a member.
- **Can be implemented in several ways**
  - Using foreign key association
  - Using a join table

One to Many Association

- **Using foreign key association**
  - In this approach, both entity will be responsible for making the relationship.
  - Employee entity should declare that relationship is One to many, and Account entity should declare that relationship from its end is many to one.



One to Many Association

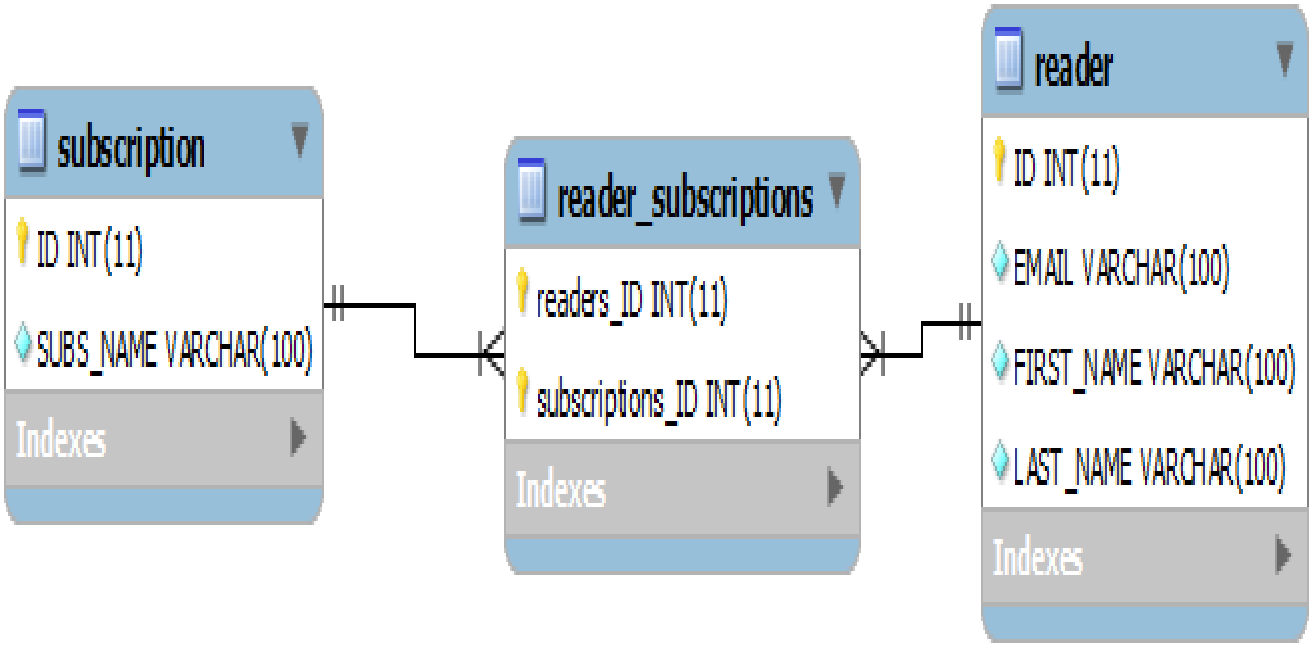

```
public class Employee
{
@OneToMany(cascade=CascadeType.ALL) @JoinColumn(name="EMPLOYEE_ID")
  private Set<Account> accounts;
}
class Account
{
@ManyToOne private
Employee employee;
}
```

Many to Many Association

- In ManytoMany unidirectional relation owner entity contains a collection of owned entities as its member.
- In M2M bidirectional relation both owner and owned contain collection of other entities as member.
- In M2M unidirectional and bidirectional relation can only be mapped with the help of a relation table.
- In M2M bidirectional relation, entity at either end can be made owner.

Mapping Continued

- @JoinTable annotation has been used to make this association.



Component Mapping

- **Component Mapping/Value type Object**
  - An object of value type has no database identity;
  - Component is a contained object that is persisted as a value type, not an entity reference.
  - Here single table would be used to keep the attributes contained inside the class variable.
  - It is used when entities are strongly related (composition relation), it is better to store them in a single table.

Mapping (Contd..)

- **@Embeddable** : Annotations is provided to embed a value type object into our Entity class.
- **Object of Entity Type** : Has its own database identity
- **Object of Value Type :** Belongs to an entity, and its persistent state is embedded in the table row of the owning entity.
- Value types don't have identifiers or identifier properties.

infogain

Introduction to HQL

Why HQL?

- Session provide methods like save,get,delete and update to perform **crud** operations.
- But what if we want to Select or Update or Insert all the records of a table based on some condition ?

- **Consider the below scenarios**
  - What if we want to select all the records of a table ?
  - What if we want to perform aggregations on columns like avg(salary) of Employees.
  - Answer to all the above queries is “HQL”.

What is HQL?

- **What is HQL?**
  - The Hibernate Query Language is an object-oriented dialect of the familiar relational query language SQL.
  - HQL is very similar to SQL except that we use Class names instead of table names and attributes of a class instead of Columns of a table .
- **HQL supports the following:**
  - The ability to apply restrictions to properties of associated objects.
  - The ability to order the results of the query.
  - Aggregation with group by, having, and aggregate functions like sum, min,and max.
  - Outer joins when retrieving multiple objects per row.

HQL (Contd..)

- Any **HQL** may consist of following elements:
  - Clauses
  - Aggregate functions
  - Subqueries

Clauses in the HQL

- from
- select
- where
- order by
- group by

- **Order by**
- ”from cust as c1 order by cust.fname“
- **Group By**
- "select sum(insurance.investementAmount), insurance.insuranceName " + "from Insurance insurance group by insurance.insuranceName";

Query interfaces

- The Query interface allows you to perform queries against the database and control how the query is executed.

- **Commonly used methods are**
  - **list() :** Return the query results as a List.
  - **iterate() :** Return the query results as an Iterator.
  - **executeUpdate() :** Execute the update or delete statement.

What is HCQL?

- **What is HCQL?**
  - It's a Criteria based query language mainly used to fetch the records based on specific search criteria.
  - HCQL can not be used to perform DDL operations like Insert,Update and Delete.
  - It can be used only for retrieving the records based on search conditions.
  - It provides the facility of pagination
  - **org.hibernate.Criteria** interface has provided several methods to add search conditions.

Most commonly used method of Criteria

- **createCriteria ()**
  - This method is used to create a new Criteria, "rooted" at the associated entity
- **public Criteria add(Criterion c)**
  - Used to add restrictions on the search results.
- **uniqueResult ()**
  - This method is used to instruct the Hibernate to fetch and return the unique records from database.
- **public List list()**
  - This method returns the list of object on which we are searching.
- **public Criteria addOrder(Order o)**
  - Used to define the ordering of result like ascending or descending.
- **public Criteria setFirstResult(int firstResult)**
- **public Criteria setMaxResult(int totalResult)**
  - These two methods are used to achieve pagination by specifying first and maximum records to be retrieved.
- **public Criteria setProjection(Projection projection)**
  - This method is used to set the projection to retrieve only specific columns in the result.

Possible restriction used in HCQL

- lt(less than)
- le(less than or equal)
- gt(greater than)
- ge(greater than or equal)
- eq(equal)
- ne(not equal)
- between
- like

Cache Management

- Hibernate caching improves the performance of the application by pooling the object in the cache.
- Cache sits between application and database.
- **First Level Cache**
  - Session object holds the first level cache data. It is enabled by default.
- **Second Level Cache**
  - SessionFactory object holds the second level cache data. The data stored in the second level cache will be available to entire application. But we need to enable it explicitly.
- **Second Level Cache implementations are provided by different vendors such as:**
  - EH (Easy Hibernate) Cache
  - Swarm Cache
  - OS Cache
  - JBoss Cache

How secondary cache works ?

- Whenever we try to load an entity , Hibernate first looks at a primary cache associated with a particular session.
- If cached entity is found in the primary cache itself then it will be returned.
- If requested entity is not found in primary cache,then hibernate looks at the second level cache.
- If requested entity is found in second level cache,then it will be returned.
- If requested entity is not found in secondary cache then database call is made to get the entity and it will be kept in both primary and secondary cache and then it will be returned.

How to enable secondary cache ?

- <!-- Enable the second-level cache -->
- **Step1 Add below configuration setting in hibernate.cfg.xml file**
- <property name="hibernate.cache.provider_class">org.hibernate.cache.HashtableCacheProvider</property>
- <property name="hibernate.cache.use_second_level_cache">true</property>
- **Add cache usage setting in hbm file or annotated class as below**
- XML file – < cache usage="read-only" / >
- Annotated class – @Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region=”employeeCache”)

Create ehcache configuration file to Define the cache property

- **Below elements can be written inside ehcache.xml file**
- **defaultCache** : Used for all the persistent classes. we can also define persistent class explicitely by using the cache element.
- **eternal** : May has two value eternal="true", we don't need to define **timeToIdleSeconds** and **timeToLiveSeconds** attributes it will be handled by hibernate internally.
- Specifying **eternal="false"** gives control to the programmer, but we need to define timeToIdleSeconds and timeToLiveSeconds attributes.
- **timeToIdleSeconds** : It defines that how many seconds object can be idle in the second level cache.
- **timeToLiveSeconds:** It defines that how many seconds object can be stored in the second level cache whether it is idle or not.
- <ehcache>
- <!-- <defaultCache maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120" timeToLiveSeconds="200" /> -->
- <cache name="empcache" eternal="true" />
- </ehcache>

infogain

Thank You