**Entità JPA**

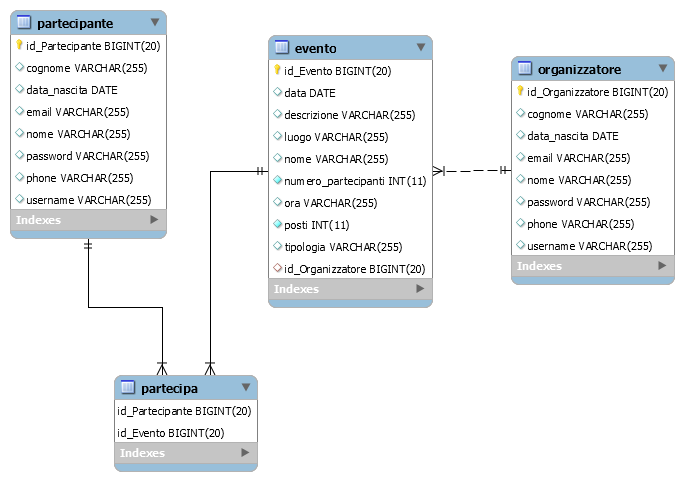


Figura , base di dati

Consideriamo la relazione esistente tra un organizzatore e l’evento organizzato, mostrata nel modello dati della figura 1, dove un organizzatore può organizzare più eventi ma un evento appartiene ad un unico organizzatore.

Utilizzando le annotazioni JPA il mapping tra le classi OrganizzatoreDb ed EventoDb si realizzeranno attraverso una annotazione @ManyToOne (dalla riga 19 alla riga 21) sulla seconda:

1. @Entity(name="EventoDb")
2. @Table(name="evento")
3. **public** **class** EventoDb {
4. @Column(name="id\_Evento")
5. @Id
6. @GeneratedValue(strategy=GenerationType.IDENTITY)
8. **private** **long** id;
9. **private** String nome;
10. **private** String luogo;
11. @Temporal(javax.persistence.TemporalType.DATE)
12. @Column(name = "data")
13. **private** Date data;
14. **private** String ora;
15. **private** **int** posti;
16. **private** String tipologia;
17. **private** String descrizione;
18. **private** **int** numero\_partecipanti;
19. @ManyToOne(fetch = FetchType.EAGER)
20. @JoinColumn(name = "id\_Organizzatore")
21. **private** OrganizzatoreDb organizzatore;
22. @ManyToMany(cascade = {CascadeType.PERSIST,CascadeType.MERGE},
23. fetch = FetchType.EAGER)
24. @JoinTable(
25. name="partecipa",
26. joinColumns=@JoinColumn(name="id\_Partecipante"),
27. inverseJoinColumns=@JoinColumn(name="id\_Evento"))
29. **private** Set<PartecipanteDb> partecipazioni=**new** HashSet<>();
30. }

Ed una @OneToMany(dalla riga 18 alla riga 24) sulla prima:

1. @Entity(name="OrganizzatoreDb")
2. @Table(name="organizzatore")
3. **public** **class** OrganizzatoreDb {
4. @Column(name="id\_Organizzatore",updatable = **false**, nullable = **false**)
5. @Id
6. @GeneratedValue(strategy=GenerationType.IDENTITY)
7. **private** **long** id;
8. **private** String nome;
9. **private** String cognome;
10. @Temporal(javax.persistence.TemporalType.DATE)
11. @Column(name = "data\_nascita")
12. **private** Date data\_nascita;
13. @Column(name="email",unique=**true**)
14. **private** String email;
15. **private** String password;
16. **private** String username;
17. **private** String phone;
18. @OneToMany(
19. mappedBy = "organizzatore",
20. cascade = CascadeType.ALL,
21. orphanRemoval = **true**,
22. fetch = FetchType.EAGER
23. )
24. **private** Set<EventoDb> eventiCreati;
26. //funzioni utili per la classe
27. **public** **void** addEvento(EventoDb ev){
28. eventiCreati.add(ev);
29. ev.setOrganizzatore(**this**);
30. }
31. **public** **void** removeEvento(EventoDb ev){
32. eventiCreati.remove(ev);
33. ev.setOrganizzatore(**null**);
34. }
35. }

Si noti che sia nella prima che nella seconda entità è stata specificata **EAGER** come strategia di fetching ma, più importante, è stata utilizzata: nella classe EventoDb, l’annotazione @JoinColumn per indicare la colonna della tabella evento che contiene il valore della primary key referenziata; nella tabella organizzatore, la proprietà di mappping(riga 19) per rendere bidirezionale l’associazione.

**Relazione Many-To-Many**

Prendiamo in considerazione la relazione esistente tra un evento ed i suio partecipanti. In questo caso un partecipante può partecipare a più eventi ed analogamente ad un evento possono partecipare più partecipanti. Una relazione di questo tipo può essere catturata esclusivamente utilizzando una join table tra le 2 entità, vedi la tabella partecipa nella figura 1.

Per mappare tutti gli eventi a cui partecipa un partecipante utilizziamo l’annotazione @ManyToMany, visibile nella classe PartecipanteDB, alla riga 22 e nella classe EventoDb alla riga 31.

1. @Entity(name="PartecipanteDb")
2. @Table(name="partecipante")
3. **public** **class** PartecipanteDb {
4. @Column(name="id\_Partecipante")
5. @Id
6. @GeneratedValue(strategy=GenerationType.IDENTITY)
7. **private** **long** id;
8. **private** String nome;

11. **private** String cognome;
12. @Temporal(javax.persistence.TemporalType.DATE)
13. @Column(name = "data\_nascita")
14. **private** Date data\_nascita;
15. @Column(name="email",unique=**true**)
16. **private** String email;
17. **private** String password;
18. **private** String username;
19. **private** String phone;

22. @ManyToMany(
23. mappedBy="partecipazioni",
24. fetch = FetchType.EAGER)
26. **private** Set<EventoDb> book=**new** HashSet<EventoDb>();
28. //funzioni utili
30. **public** **void** addBook(EventoDb ev){
31. book.add(ev);
32. ev.getPartecipazioni().add(**this**);
34. }
35. **public** **void** removeBook(EventoDb ev){
36. book.remove(ev);
37. ev.getPartecipazioni().remove(**this**);
38. }
39. }
40. @ManyToMany(cascade = {CascadeType.PERSIST,CascadeType.MERGE},
41. fetch = FetchType.EAGER)
42. @JoinTable(
43. name="partecipa",
44. joinColumns=@JoinColumn(name="id\_Partecipante"),
45. inverseJoinColumns=@JoinColumn(name="id\_Evento"))
47. **private** Set<PartecipanteDb> partecipazioni=**new** HashSet<>();

Si noti che la tabella partecipa non ha un’entità nel progetto ma è semplicemente referenziata dall’annotazione @JoinTable (riga 34) che serve a dichiarare il mapping tra le entità EventoDb e PartecipanteDb.

**Esempio di operazioni CRUD (create, read, update, delete)**

**Create**

Per memorizzare un oggetto creato sul database è necessario utilizzare l’oggetto Entity Manager per iniziare una transazione, rendere effettivo il caricamento ed infine chiudere la connessione. Nell’esempio seguente il codice usato per l’inserimento di un nuovo partecipante o di un nuovo organizzatore nel DB.

1. **public** **static** **int** registrazione(OrganizzatoreDb organizzatore,PartecipanteDb partecipante){
2. **int** errore=1;
3. **if**(((organizzatore==**null**)&&(partecipante==**null**))||((organizzatore!=**null**)&&(partecipante!=**null**)))
4. {
5. System.err.println("parametri non validi riprova");
6. **return** 0;
8. }
9. **try**{
10. creaConnessione();
11. entityManager.getTransaction().begin();
12. **if**(organizzatore!=**null**)
13. entityManager.persist(organizzatore);
14. **else**
15. entityManager.persist(partecipante);
17. entityManager.getTransaction().commit();
18. }
19. **catch**(PersistenceException pe)
20. {
21. pe.printStackTrace();
22. errore=0;
23. }
24. **catch**(Exception e)
25. {
26. e.printStackTrace();
27. System.out.println("Errore durante la transizione riprova!");
28. errore=0;
29. }
30. **finally**
31. {
32. chiudiConnessione();
33. }
34. **return** errore;
35. }

**Read**

Eseguiamo un interrogazione al DB tramite la chiamata del metodo getResultList() dell’Entity Manager che restituisce una lista oggetti. Nell’esempio seguente il metodo restituisce tutti gli eventi ancora disponibili e al quale il partecipante non si è ancora iscritto.

1. **public** **static** ArrayList<Evento> ricercaEventi(PartecipanteDb partecipante, String Citta){
3. String sql;
4. List<EventoDb> listaEventi;
5. ArrayList<Evento> ev=**new** ArrayList<>();
7. **try**{
8. entityManager = factory.createEntityManager();
9. entityManager.getTransaction().begin();
10. **if**(Citta.equals("")){ //Caso in cui un partecipante vuole visualizzare gli eventi a cui è iscritto
11. sql="select e from EventoDb e where e.numero\_partecipanti<e.posti and  e.data>=current\_date() and e.id not in (select d.id from PartecipanteDb p join p.book d  where  p.id="+partecipante.getId()+")";
13. } **else**{ //Caso in cui un partecipante vuole visualizzare tutti gli eventi
14. sql="select e from EventoDb e where e.numero\_partecipanti<e.posti and e.luogo='"+Citta+"' and e.data>=current\_date() and e.id not in (select d.id from PartecipanteDb p join p.book d  where  p.id="+partecipante.getId()+")";
15. }
17. TypedQuery<EventoDb> query= entityManager.createQuery(sql, EventoDb.**class**);
18. listaEventi = query.getResultList();
20. **for** (EventoDb evento : listaEventi) {
21. ev.add( **new** Evento((**int**)evento.getId(), evento.getNome(), evento.getLuogo(), evento.getData(),
22. evento.getOra(), evento.getPosti(), evento.getTipologia(), evento.getDescrizione(),
23. (**int**)evento.getOrganizzatore().getId(), evento.getNumero\_partecipanti()));
24. }
25. entityManager.getTransaction().commit();
26. } **catch**(Exception ex){
27. ex.printStackTrace();
28. System.out.println("A problem occured in searching events!");
29. } **finally** {
30. entityManager.close();
31. }
32. **return** ev;
33. }

**Update**

Nel seguente esempio verranno modificati gli attributi di un partecipante, tramite l’apertura della transazione, il merge dell’oggetto partecipanteDb della chiusura della transazione.

1. **public** **static** **int** modificaDati(PartecipanteDb partecipante) {
2. **int** errore=2;
3. **try**{
4. entityManager = factory.createEntityManager();
5. entityManager.getTransaction().begin();
6. entityManager.merge(partecipante);
7. entityManager.getTransaction().commit();
8. errore=1;
10. } **catch**(Exception ex){
11. ex.printStackTrace();
12. System.out.println("A problem occured in insert events!");
13. errore=2;
14. }**finally**{
15. entityManager.close();
16. }
17. **return** errore;
18. }

**Delete**

Cancellazione dell’evento sfruttando il metodo remove dell’oggetto Entity Manager.

1. **public** **static** **int** eliminaEvento(**long** id) {
2. **int** errore = 1;
3. **try**{
4. entityManager = factory.createEntityManager();
5. entityManager.getTransaction().begin();
6. EventoDb ev=entityManager.find(EventoDb.**class**, id);
7. **for**(Iterator<PartecipanteDb> it=ev.getPartecipazioni().iterator();it.hasNext();){
8. PartecipanteDb p=it.next();
9. it.remove();
10. ev.removePartecipante(p);
11. }
12. ev.removeOrganizzatore();
13. entityManager.remove(ev);
14. entityManager.getTransaction().commit();
15. System.out.println("EVENTO Cancellato");
17. } **catch**(Exception ex){
18. ex.printStackTrace();
19. entityManager.getTransaction().rollback();
20. System.out.println("A problem occured in  delete an event!");
21. errore = 0;
22. }
23. **finally**{
24. entityManager.close();
25. }
26. **return** errore;
27. }