Trainer: Nilesh Ghule

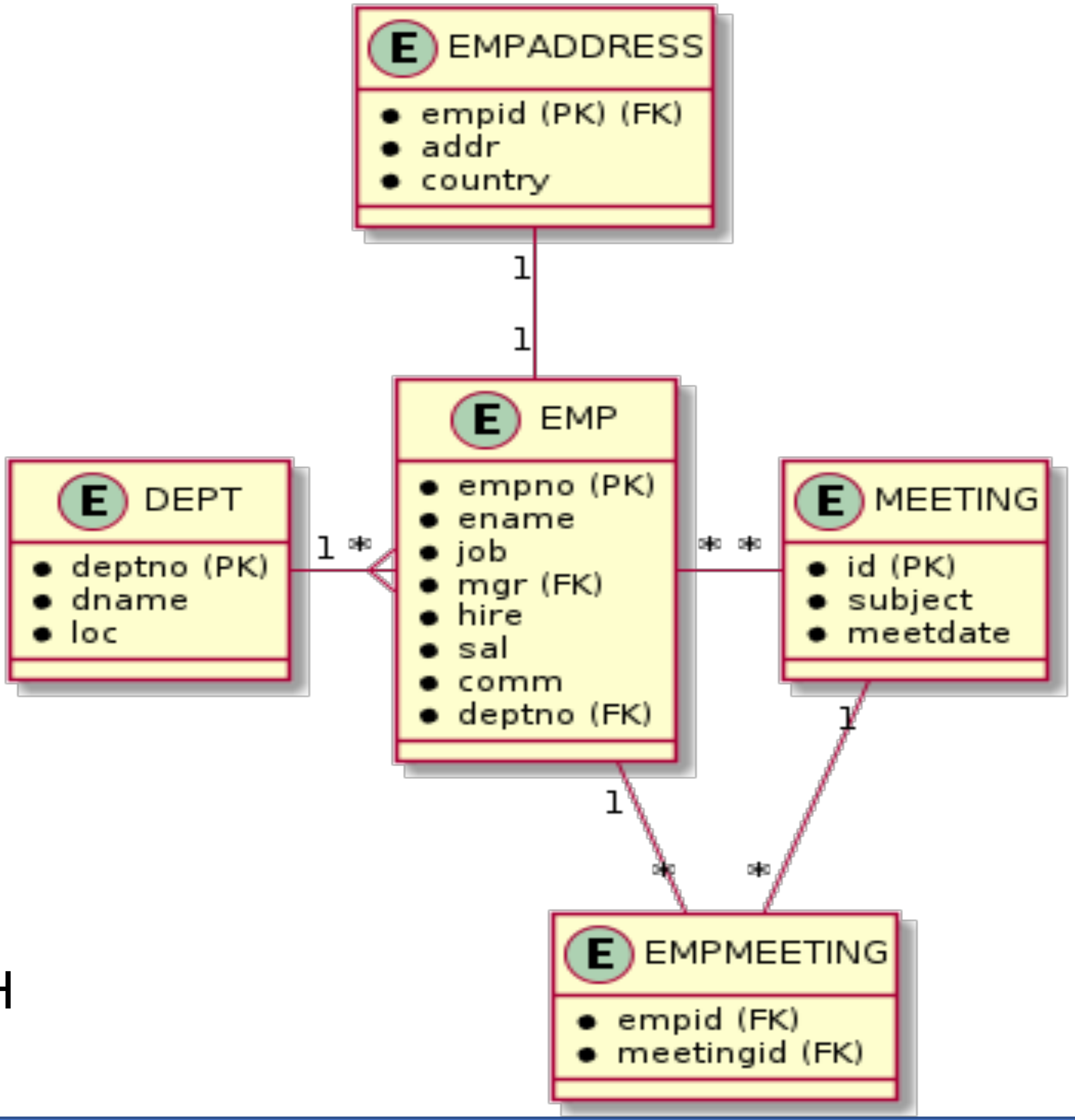
Wake up from Hibernate, Spring up!!!

Agenda

- Hibernate relations
  - @OneToMany ✓
  - @ManyToOne ✓
  - @ManyToMany ✓
  - @OneToOne ✓
- HQL Joins ✓
- Hibernate Inheritance ✓
  - @MappedSuperclass ✓
  - @Inheritance ✓
- Forward engineering vs Reverse engineering

Hibernate Relations (associations)

- Hibernate represents RDBMS table relations.
  - OneToOne ✓
  - OneToMany ✓
  - ManyToOne ✓
  - ManyToMany ✓
- OneToMany & ManyToOne represent parent-child relation between tables.
- Primary key of parent table is mapped to foreign key of child table.
- FetchType
  - Lazy or Eager
- CascadeType
  - PERSIST, MERGE, DETACH, REMOVE, REFRESH



@OneToMany (uni-directional)

- A Dept has Many Emp.
  - mappedBy – foreign key field in Emp table (that reference to primary key of Dept table). → ~~Column~~ @Id
  - FetchType
    - LAZY: Fetch Dept only (simple SELECT query) and fetch Emp only when empList is accessed (simple SELECT query with WHERE clause on deptno)
    - EAGER: Fetch Dept & Emp data in single query (OUTER JOIN query)
  - CascadeType → JPA
    - ✓ PERSIST: insert Emp in list while inserting Dept (persist())
    - ✓ REMOVE: delete Emp in dept while deleting Dept (remove())
    - ✓ DETACH: remove Emp in dept from session while removing Dept from session (detach())
    - ✓ REFRESH: re-select Emp in dept while re-selecting Dept (refresh())
    - ✓ MERGE: add Emp in dept ~~intion~~ into session while adding Dept into session (merge())

ALL

$$\begin{tikzcd}
\text{Transient} \arrow[d] \\
\text{Persistent} \arrow[d] \arrow[dd, bend left] \\
\text{Removed} \\
\text{Detached} \arrow[uu, bend left]
\end{tikzcd}$$

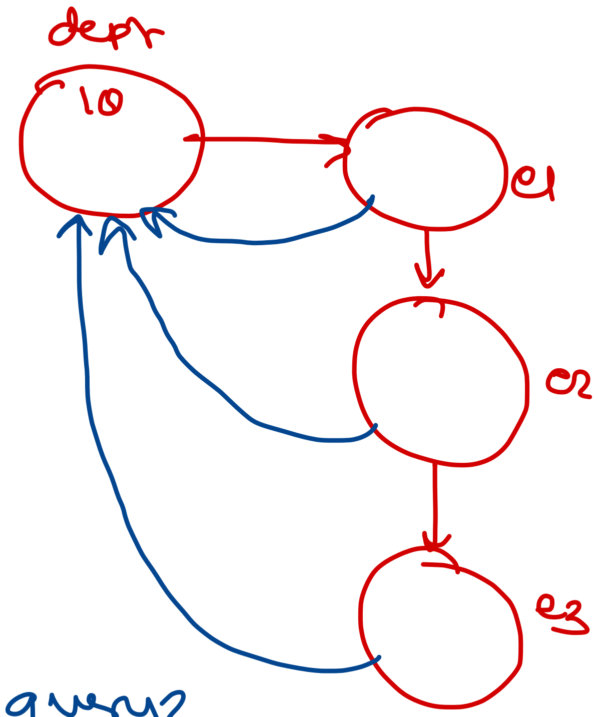
```
@Entity @Table(name="dept")
class Dept {
    @Id
    @Column private int deptno;
    @Column private String dname;
    @Column private String loc;
    @OneToMany(mappedBy="deptno")
    private List<Emp> empList;
    // ...
}
```

FK field

```
@Entity @Table(name="emp")
class Emp {
    @Id
    @Column private int empno;
    @Column private String ename;
    @Column private double sal;
    @Column private int deptno;
    // ...
}
```

@ManyToOne (uni-directional)

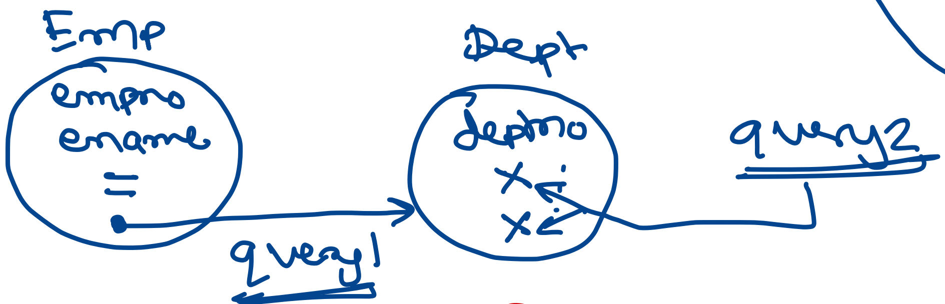
- Many Emp can have same Dept.
  - FetchType – LAZY or EAGER*
  - CascadeType - PERSIST, MERGE, DETACH, REMOVE, REFRESH
  - @JoinColumn is used along with @ManyToOne to specify foreign key column in EMP table (that reference to primary key of DEPT table).



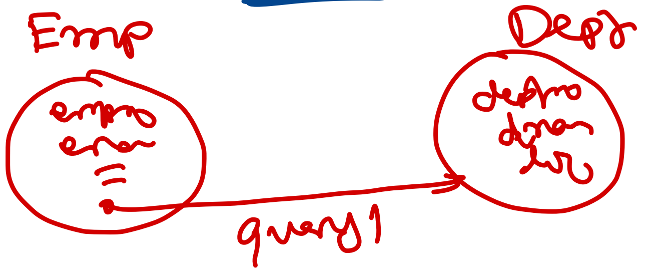
```
@Entity @Table(name="dept")
class Dept {
    @Id
    @Column private int deptno;
    @Column private String dname;
    @Column private String loc;
    // ...
}

@Entity @Table(name="emp")
class Emp {
    @Id
    @Column private int empno;
    @Column private String ename;
    @Column private double sal;
    ~~@Column private int deptno;~~ ×
    @ManyToOne → fetch/cascade.
    @JoinColumn(name="deptno")
    private Dept dept;
    // ...
}
```

FK column

Lazy



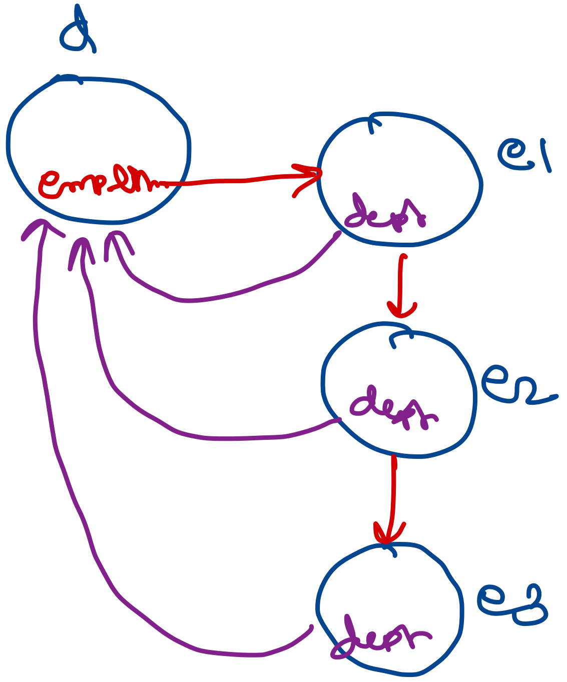
Eager



e.dept.deptno

@OneToMany and @ManyToOne (bi-directional)

- A Dept have many Emps.
- Many Emp can have same Dept.
- @ManyToOne in Emp class
  - Use @JoinColumn to speficy FK column in EMP table.
- @OneToMany in Dept class
  - Use mappedBy to specify FK field in Emp class – now declared as Dept object.
  - FK value is taken from inner Dept object's @Id field.

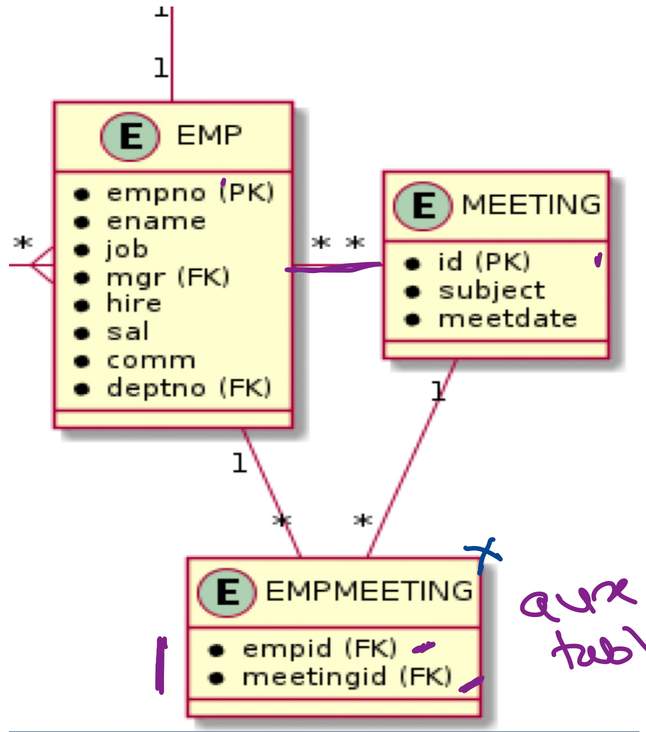


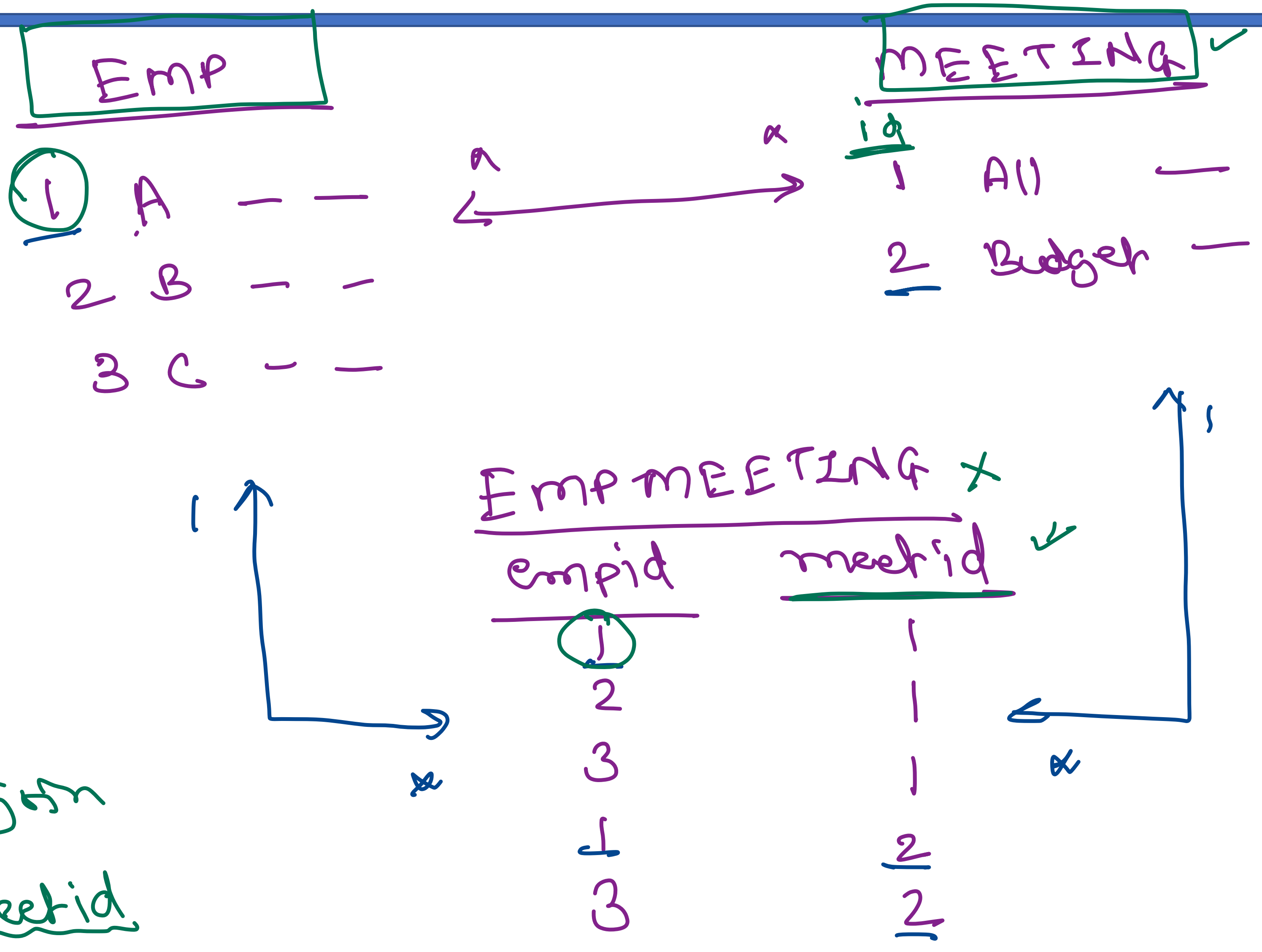
```
@Entity @Table(name=”DEPT”)
class Dept {
   @Id
   @Column private int deptno;
   @Column private String dname;
   @Column private String loc;
   @OneToMany(mappedBy="dept")
   private List<Emp> empList;
}

@Entity @Table(name=”EMP”)
class Emp {
   @Id
   @Column private int empno;
   @Column private String ename;
   @Column private double sal;
   @Column private int deptno;
   @ManyToOne
   @JoinColumn(name=”deptno”)
   private Dept dept;
}
```

@ManyToMany (bi-directional)

- One Emp can have many Meetings.
- One Meeting will have many Emps.
- Many-to-many relation is established into two tables via an additional table (auxilary table).
- The EMP_MEETING table holds FK of both tables to establish the relation.
- In first class (e.g. Emp) use @ManyToMany along with @JoinTable (refering auxilary table & FK column in it).
  - joinColumn – first table's FK in aux table
  - inverseJoinColumn – second table's FK in aux table
- In second class (e.g. Meeting) use @ManyToMany with mappedBy to setup bi-directional relation.

```
class Emp {
    @Id
    @Column private int empno;
    @Column private String ename;
    @ManyToMany
    @JoinTable(name = "EMPMEETING",
      joinColumns = {@JoinColumn (name="EMPID")},
     inverseJoinColumns = {@JoinColumn (name="MEETINGID")} )
            private List<Meeting> meetingList;
}
class Meeting {
            @Id
            @Column private int id;
            @Column private String subject;
            @ManyToMany(mappedBy="meetingList")
            private List<Emp> empList;
}
```



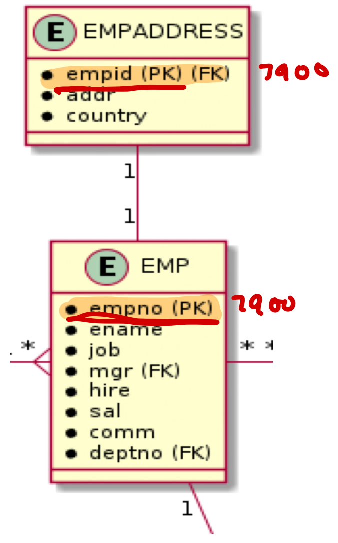


empid	meetid
1	1
2	1
3	1
1	2
3	2

select ___ from
meeting m inner join
empmeeting em
on m.id = em.meetid
where em.empid = ?

@OneToOne (bi-directional)

- One Emp have one Address.
- If both tables have same primary key, then use @OneToOne along along with @PrimaryKeyJoinColumn. Use @OneToMany with mappedBy in second class to setup bidirectional relation.
- If a table conatins FK for another table use @OneToOne with @JoinColumn.

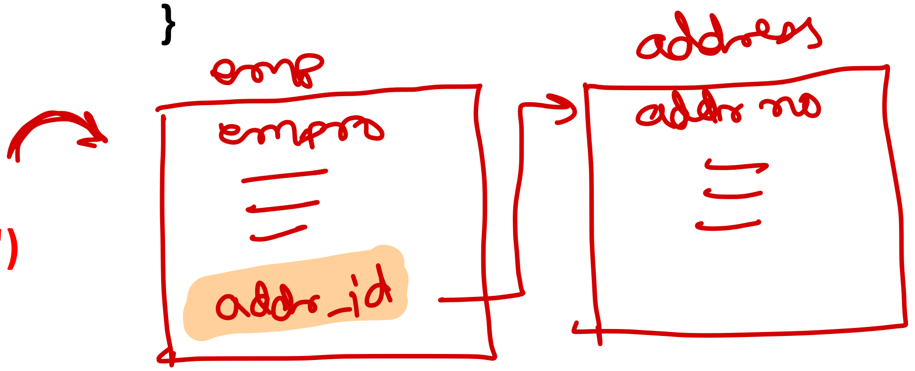


```
class Emp {
    @Id
    @Column private int empno;
    @OneToOne
    @PrimaryKeyJoinColumn
     private Address addr;
}
class Address {
    @Id
    @Column private int empid;
    @Column private String country;
    @OneToOne(mappedBy = "addr")
    private Emp emp;
}
```

```
class Emp {
    @Id
    @Column private int empno;
    @OneToOne
    @JoinColumn(name="addr_id")
    private Address addr;
}
```

FK Column

```
class Address {
    @Id
    @Column private int id;
    @Column private String country;
    @OneToOne(mappedBy = "addr")
    private Emp emp;
}
```



SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>