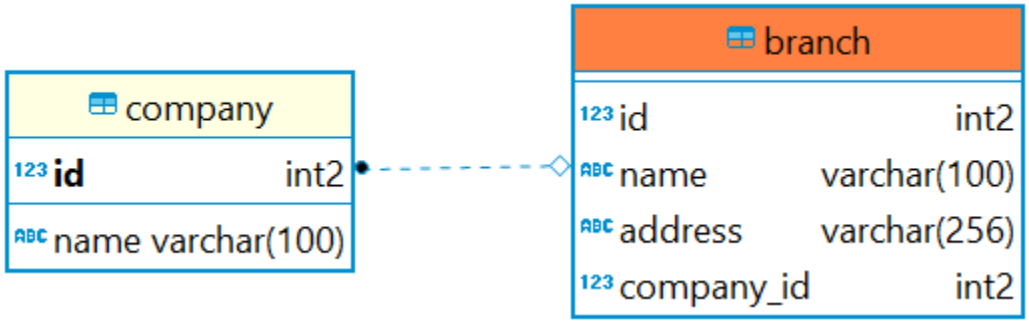
JPA 2.2

Java Persistence API (JPA):

- Standard API for object-relational mapping (ORM)
- Query objects stored in the underlying database using Java Persistence Query Language (JPQL)
- Is a Java application programming interface specification. Famous implementations: ***Hibernate***, *EclipseLink, Toplink, OpenJPA*

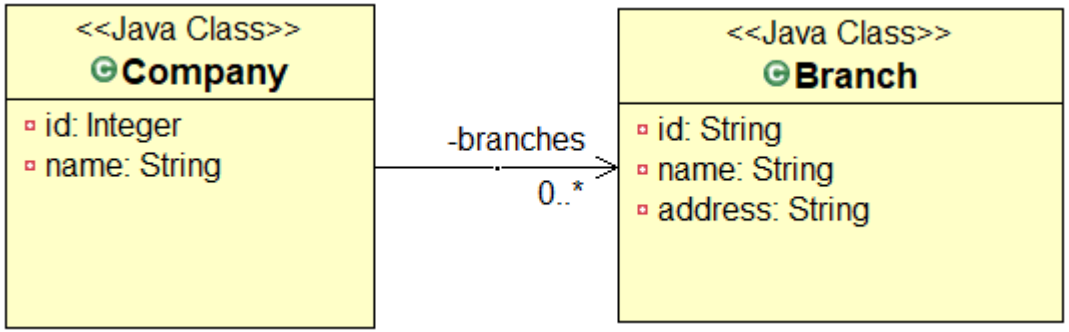
Back To Past

- Most of the data that our applications manipulate has to be stored in databases, retrieved, and analyzed.
- Relational databases store data in tables made of rows and columns
- We write SQL statements INSERT, SELECT, UPDATE, DELETE to manipulate data from / to database. As a consequence, code is complicated and difficult to maintain.

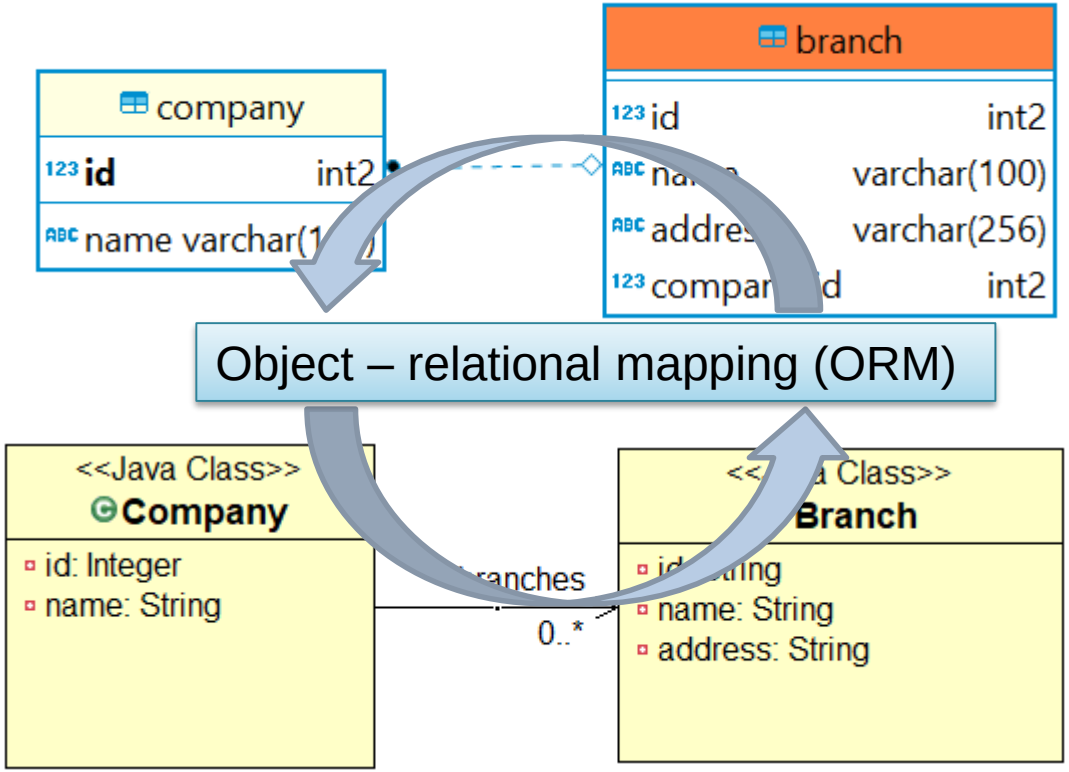

company
id int2
name varchar(100)
branch
id int2
name varchar(100)
address varchar(256)
company_id int2

Back To Past

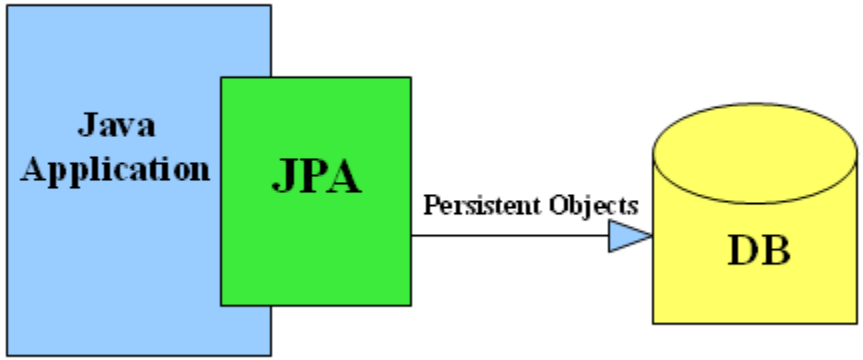
- In Java, data is stored in objects that are instances of classes
- Objects inherit from others or have collections of other objects.



How to bring the world of database and objects together?



JPA



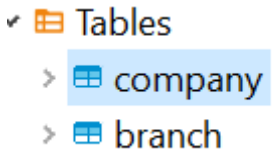
Topics are addressed

- Entity
- Embeddable class
- Object-Relational Mapping
- Inheritance mapping
- Entity Manager
- Querying Entities
- Persistence Unit
- Entity Life Cycle and Callbacks
- Controlling Concurrent Access to Entity Data with Locking
- Troubleshooting

Entity

- When an object mapped to a relational database (persisting object or querying object). It's called "entity"
- Entities are objects that live shortly in memory and persistently in a database

```java
/**
 * the company entity
 */
@Entity
@Table(name = "company")
public class Company {

    @Id
    private Integer id;

    @Column(name = "name", length = 100, nullable = false)
    private String name;

    @OneToMany
    private List<Branch> branches;
}
```



Entity

Characteristics:

- Represent a table in database
- Each instance is respective to a row in the table
- It’s annotated as @Entity
- Has a public or protected **no-arg constructor**
- Field mapping: @Basic (optional, lazy), @Column (column definition)

An example

```
@Entity
@Table(name = "tank_type")
public class TankTypeEntity implements Serializable{

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "type_name", unique = true, nullable = false, length = 10)
    private String name;

    @Column(name = "tank_weight", nullable = false)
    private Double tankWeight;

    @Column(name = "net_weight", nullable = false)
    private Double netWeight;

    public TankTypeEntity() {
    }

    // getters and setters
```

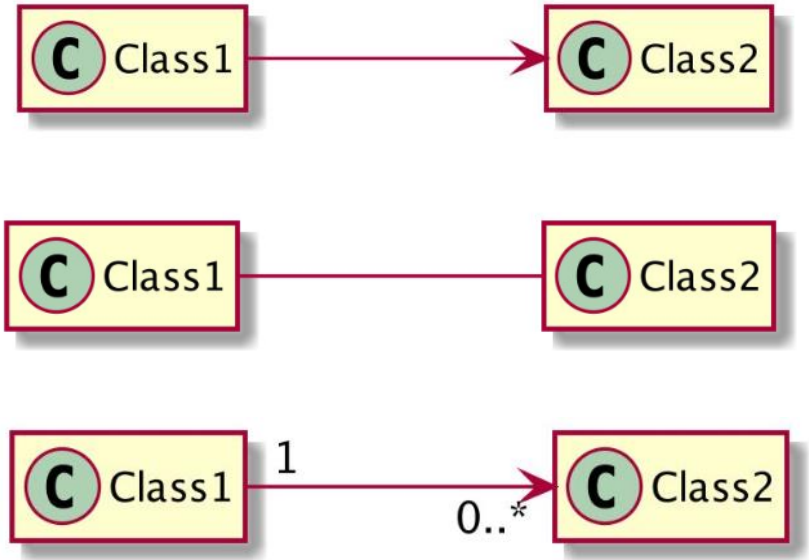
Entity-related annotations for a column

- @Basic
- @Column
- @Transient
- @Enumerated
- @ElementCollection (advance)
- ….

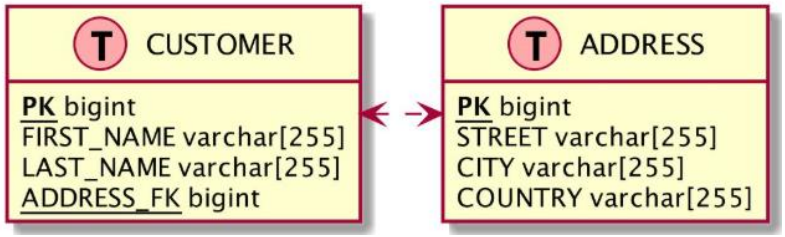
Mapping relationships

Concepts

Relationships in Objects



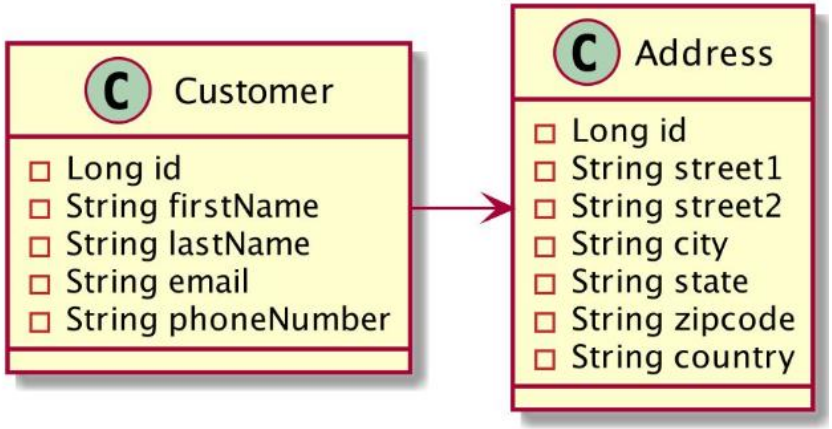
• Relationships in db



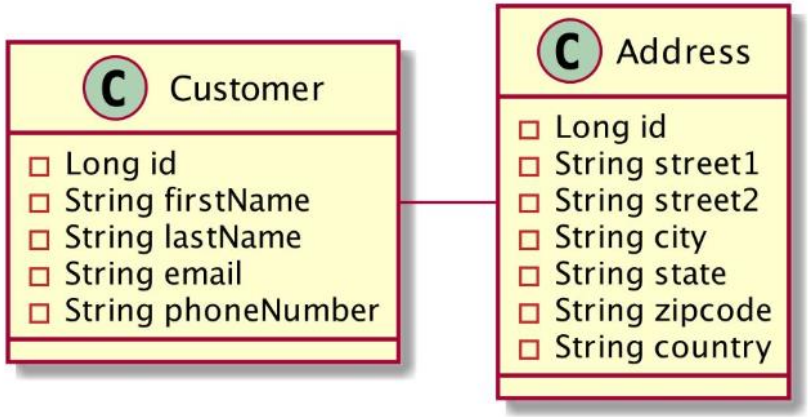
Mapping relationships

Entity relationships

Unidirectional relationship



Bidirectional relationship



Mapping relationships

Entity relationships

- `@oneToOne`
- `@oneToMany`
- `@manyToOne`
- `@manyToMany`

Mapping relationships

An example of @oneToOne

```java
@Entity
@Table(name = "company_address")
public class CompanyAddress {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @OneToOne(fetch = FetchType.LAZY)
    @MapsId
    private Company company;

    @Column(name = "country", length = 20)
    private String country;

    @Column(name = "province_or_city", length = 50)
    private String provinceOrCity;

    @Column(name = "district")
    private String district;

    @Column(name = "houseNr", columnDefinition = "text")
    private String houseNr;
}
```

Mapping relationships

An example of @OneToMany

```
20 @Entity
21 @Table(name = "company")
22 public class Company {
23
24     @Id
25     private Integer id;
26
27     @Column(name = "name", length = 100, nullable = false)
28     private String name;
29
30     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
31     @JoinColumn(name = "company_id")
32     private List<Employee> employees;
33 }
34
```

Mapping relationships

An example of @ManyToOne

```
36 public @Data class Employee {
37
38
39     @Id
40     @GeneratedValue(strategy = GenerationType.AUTO)
41     private Integer id;
42
43     @Column(name = "name", length = 50, nullable = false)
44     private String name;
45
46     @Column(name = "dob")
47     @Temporal(TemporalType.DATE)
48     private Date dateOfBirth;
49
50     @ManyToOne(fetch = FetchType.LAZY)
51     private Company workingAt;
```

Mapping relationships

Bi-directional relationship

```java
public @Data class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;


    @Column(name = "name", length = 50, nullable = false)
    private String name;


    @Column(name = "dob")
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "company_id")
    private Company workingAt;
```

```java
@Entity
@Table(name = "company")
public class Company {

    @Id
    private Integer id;

    @Column(name = "name", length = 100, nullable = false)
    private String name;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "workingAt")
    private List<Employee> employees = new ArrayList<>();

}
```

Fetching relationships

Default strategy

Annotation	Default Fetching Strategy
@OneToOne	EAGER
@ManyToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

Mapping relationships

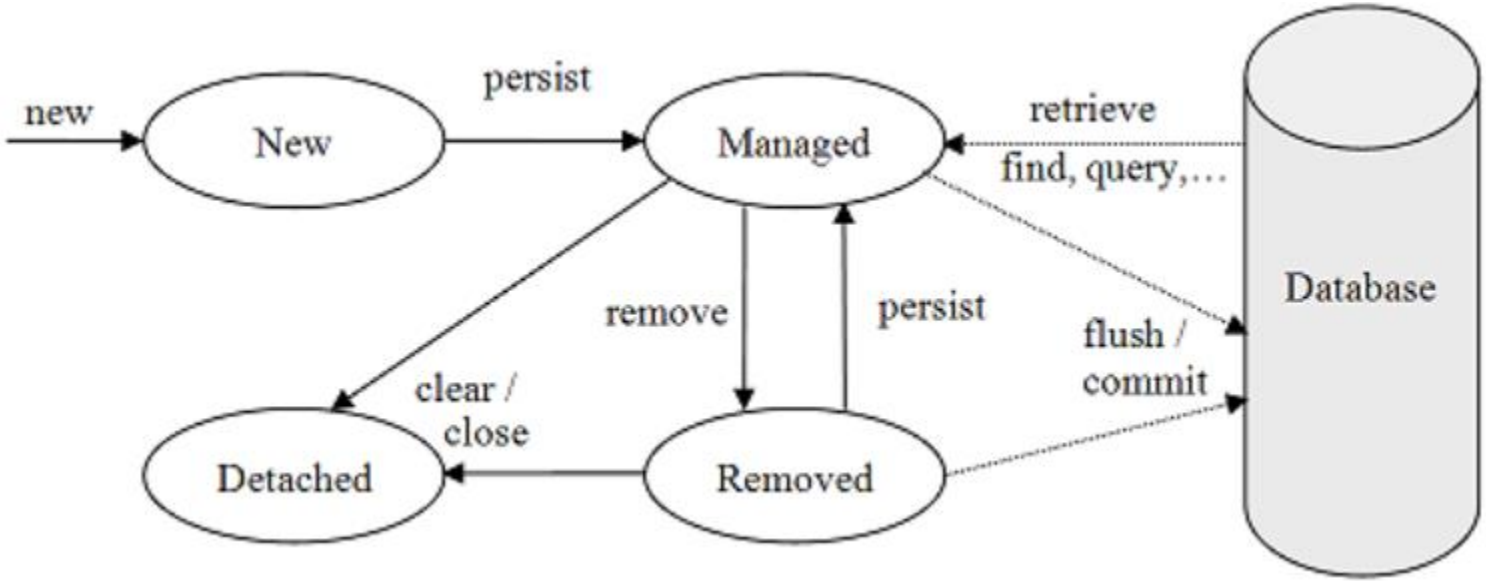
An example of @ManyToMany

```java
@Entity
@Table(name = "project")
public class Project {

    @Id
    private Integer id;

    @Column(name = "pro_name", length = 100, nullable = false)
    private String name;

    @ManyToMany(cascade = { CascadeType.PERSIST, CascadeType.MERGE})
    @JoinTable(name = "project_team",
            joinColumns = @JoinColumn(name = "project_id"),
            inverseJoinColumns = @JoinColumn(name = "team_id")
        )
    private List<Team> involvedTeams = new ArrayList<>();
}
```

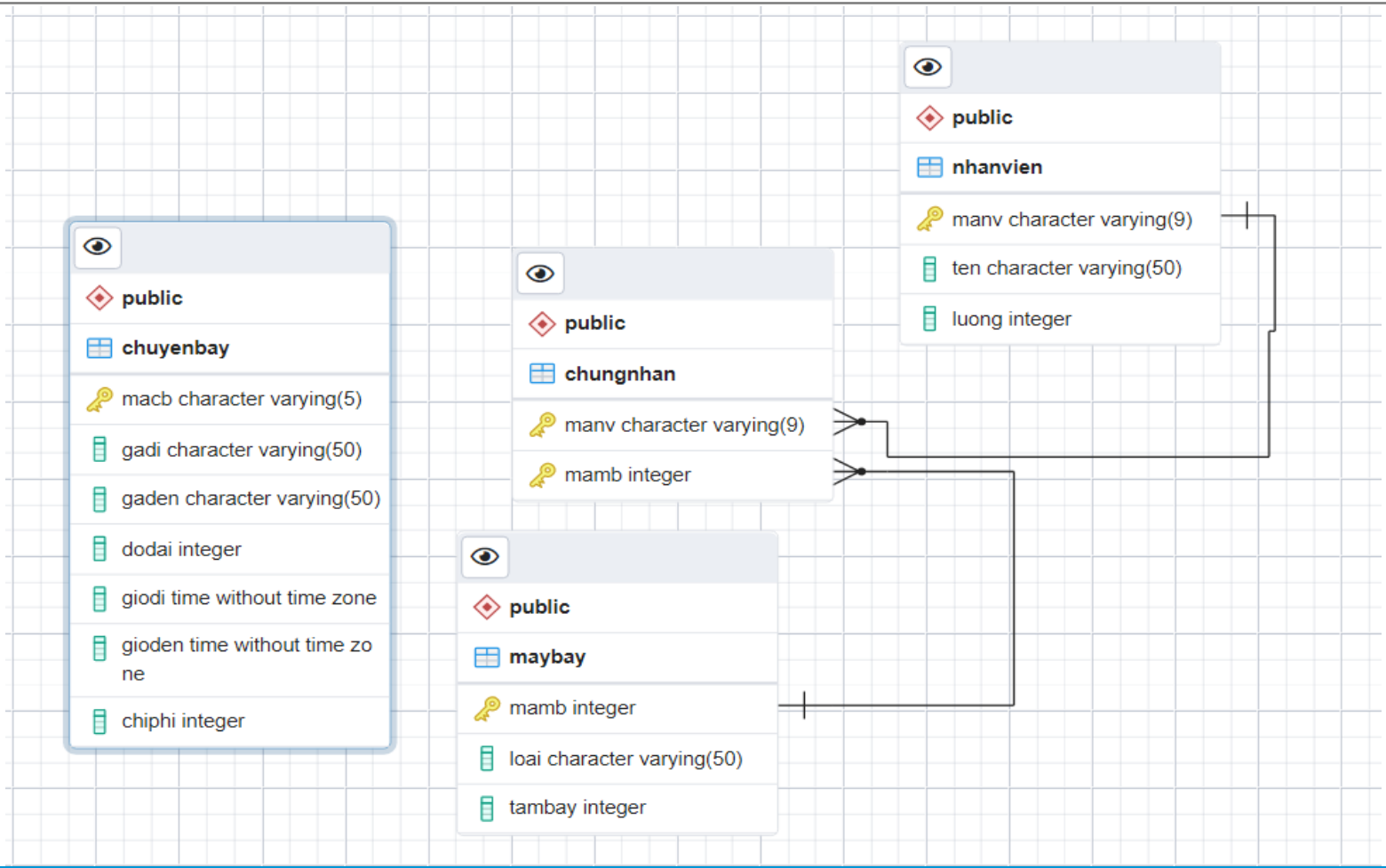
```java
@Entity
@Table(name = "team")
public class Team {

    @Id
    private Integer id;

    @Column(name = "team_name", length = 100, nullable = false)
    private String name;

    @ManyToMany(mappedBy = "involvedTeams")
    private List<Project> projects = new ArrayList<>();
}
```

Entity Manager

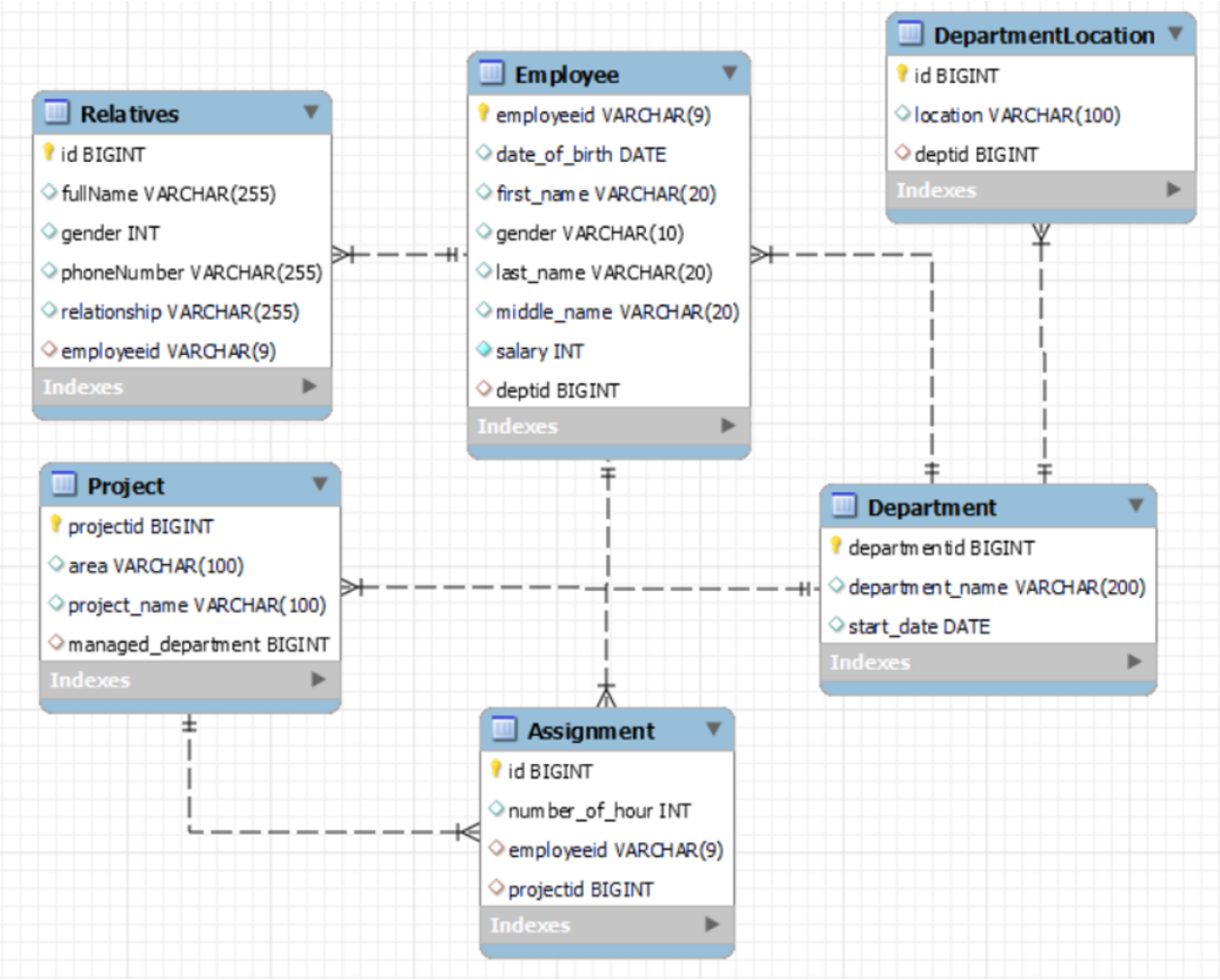
Entity lifecycle



ORM exercises



ORM exercises (cont.)



Entity Manager

- Manage entities
- Container-managed entity manager:

```
@PersistenceContext EntityManager em;
```

- Application-managed entity manager
- Some APIs: `createQuery, persist, find, merge, remove, detach`

Entity manager

APIs

```
// Persists, merges, removes and finds an entity to/from the database
public void persist(Object entity);
public <T> T merge(T entity);
public void remove(Object entity);
public <T> T find(Class<T> entityClass, Object primaryKey);
public <T> T getReference(Class<T> entityClass, Object primaryKey);

// Refreshes the state of the entity from the database, overwriting any changes
made
public void refresh(Object entity);
public void refresh(Object entity, LockModeType lockMode);

// Synchronises the persistence context to the underlying database
public void flush();
public void setFlushMode(FlushModeType flushMode);
public FlushModeType getFlushMode();
```

Entity manager

APIs

```
// Creates an instance of Query or TypedQuery for executing a JPQL statement
public Query createQuery(String qlString);
public <T> TypedQuery<T> createQuery(String qlString, Class<T> resultClass);

// Creates an instance of Query or TypedQuery for executing a named query
public Query createNamedQuery(String name);
public <T> TypedQuery<T> createNamedQuery(String name, Class<T> resultClass);

// Creates an instance of Query for executing a native SQL query
public Query createNativeQuery(String sqlString);
public Query createNativeQuery(String sqlString, Class resultClass);
public Query createNativeQuery(String sqlString, String resultSetMapping);

// Creates a StoredProcedureQuery for executing a stored procedure in the
latabase
public StoredProcedureQuery createNamedStoredProcedureQuery(String name);
public StoredProcedureQuery createStoredProcedureQuery(String procedureName);
public StoredProcedureQuery createStoredProcedureQuery(
  String procedureName, Class... resultClasses);
public StoredProcedureQuery createStoredProcedureQuery(
  String procedureName, String... resultSetMappings);
```

Query Entities

JPQL

- Similar to SQL in terms of syntax
- Works with Java classes and instances
- Returned entities are in managed status
- @NameQuery can be defined in Entity class
- To query: use EntityManager.createQuery or EntityManager.createNamedQuery

Spring Data - JPA repositories

- 1. Method names
- 2. JPA Named Queries
- 3. Using @Query
- 4. Native queries
- 5. Named Native Query

https://docs.spring.io/spring-data/jpa/docs/1.5.0.RELEASE/reference/html/jpa.repositories.html

@Query return DTO


```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class EmployeeDto {
    private String firstName;
    private String departmentName;
}
```

```java
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
    @Query("SELECT new com.axonactive.demo.service.dto.EmployeeDto(e.firstName, d.name) " +
            "FROM Employee e, Department d WHERE e.department.id = d.id and d.id = :deptid ")
    List<EmployeeDto> getEmployeeInADepartment(@Param("deptid") Integer deptid);

}
```

@NamedNativeQuery return DTO

AXON ACTIVE
www.axon.vn

```
@Entity
@Table(name = "comp_department")
@SqlResultSetMapping(
        name="DepartmentEmployeeStatistics",
        classes={
                @ConstructorResult(
                        targetClass = com.axonactive.demo.service.dto.DepartmentStatisticsDto.class,
                        columns={
                                @ColumnResult(name="departmentName", type = String.class),
                                @ColumnResult(name="numberOfEmployee", type = Long.class)})})
@NamedNativeQuery(
        name = Department.COUNT_EMPLOYEES_IN_DEPARTMENT,
        query = "SELECT d.name as departmentName, count(e.id) as numberOfEmployee " +
                "FROM comp_department d left join employee e on d.id = e.dept_id " +
                "GROUP BY d.name ORDER BY d.name",
        resultSetMapping = "DepartmentEmployeeStatistics")
public class Department {
    public static final String COUNT_EMPLOYEES_IN_DEPARTMENT = "Department.countEmployeesInDepartments";
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false)
```

@NamedNativeQuery return DTO


```
import java.util.List;

@Repository
public interface DepartmentRepository extends JpaRepository<Department,Integer> {

    @Query(nativeQuery=true)
    List<DepartmentStatisticsDto> countEmployeesInDepartments();
```

@NamedQuery


```
@Entity
@Table(name = "edu_course")
@XmlRootElement
@NamedQueries({
  @NamedQuery(name = "EduCourse.findByCompanyNameAndCourseName", query = "SELECT e FROM EduCourse e "
      + "JOIN e.companyId c WHERE e.courseName = :courseName AND c.companyName = :companyName")})
public class EduCourse implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Basic(optional = false)
  @Column(name = "id")
  private Integer id;

  @JoinColumn(name = "company_id", referencedColumnName = "id")
  @ManyToOne
  private EduCompany companyId;

  public List<EduCourse> find(String courseName, String companyName) {
    Query query = em.createNamedQuery("EduCourse.findByCompanyNameAndCourseName");
    query.setParameter("courseName", courseName);
    query.setParameter("companyName", companyName);
    return query.getResultList();
  }
```

JPA Query Exercises

1. Method name (20)

https://docs.spring.io/spring-data/jpa/docs/1.5.0.RELEASE/reference/html/jpa.repositories.html

1. @Query (15)
2. NamedQuery (15)

Using HangKhong + Company ERD