Name: Win Ei Khaing StudentID: 613403

SCI: 4
Total: 100 -- Excellent!

Theory Section

A. [3 pts] What are the 4 states of the entity life cycle?

3 Transient, Managed, Detached, Removed

B. [3 pts] Why are surrogate keys preferred over natural keys?

3 - Surrogate keys don't have meaning in business domain while natural keys do. So it can't be duplicates. (unique)
- Surrogate keys can be generated by database without null value (constant, required)

C. [3 pts] Explain what a sequence is in a database:

3 @Id
@GeneratedValue(strategy = GenerationType.SEQUENCE(name = "Test_Sequence"))
For that case, the database has sequence called Test_Sequence. By doing that we can create PK for multiple tables. So, if we join them, there won't be any duplicate ID

2 uni-directional association has owning-side & race condition problems while bi-directional association does not. only race if both onto same FK

D. [3 pts] Explain the difference between a bi-directional association and two uni-directional associations. At @OneToMany and @ManyToOne,

3 If we didn't put mappedBy, it will create 2 uni-directional association (means one JoinColumn association from owning side and JoinTable association from non-owning side)
If we put mappedBy on @OneToMany (many side) side, it will create only JoinColumn association from owning side.

E. [3 pts] What does entitManager.flush() do?

3 It pushes all updates in cache to database.
Like before query or when transaction commits.

F. [3 pts] What does the 'Extra' @LazyCollection do in terms of Hibernate optimization?

3 If we write @LazyCollection(LazyCollectionOption.EXTRA) on a collection, when we want count of that collection, Hibernate will run SELECT COUNT(..)... for query. If not, it will retrieve all data for collecti From db and then calculate count the ro

G. [3 pts] Explain how a version column can fix the Lost Update problem

3 JPA can solve Lost Update problem with using version column (Optimistic concurrency).
So, 2 transactions update a same data, at the same time, after one is successful, version column value was changed. So, the second transaction will got exception, failing update.

H. [3 pts]Explain what a (XA) global transaction is:

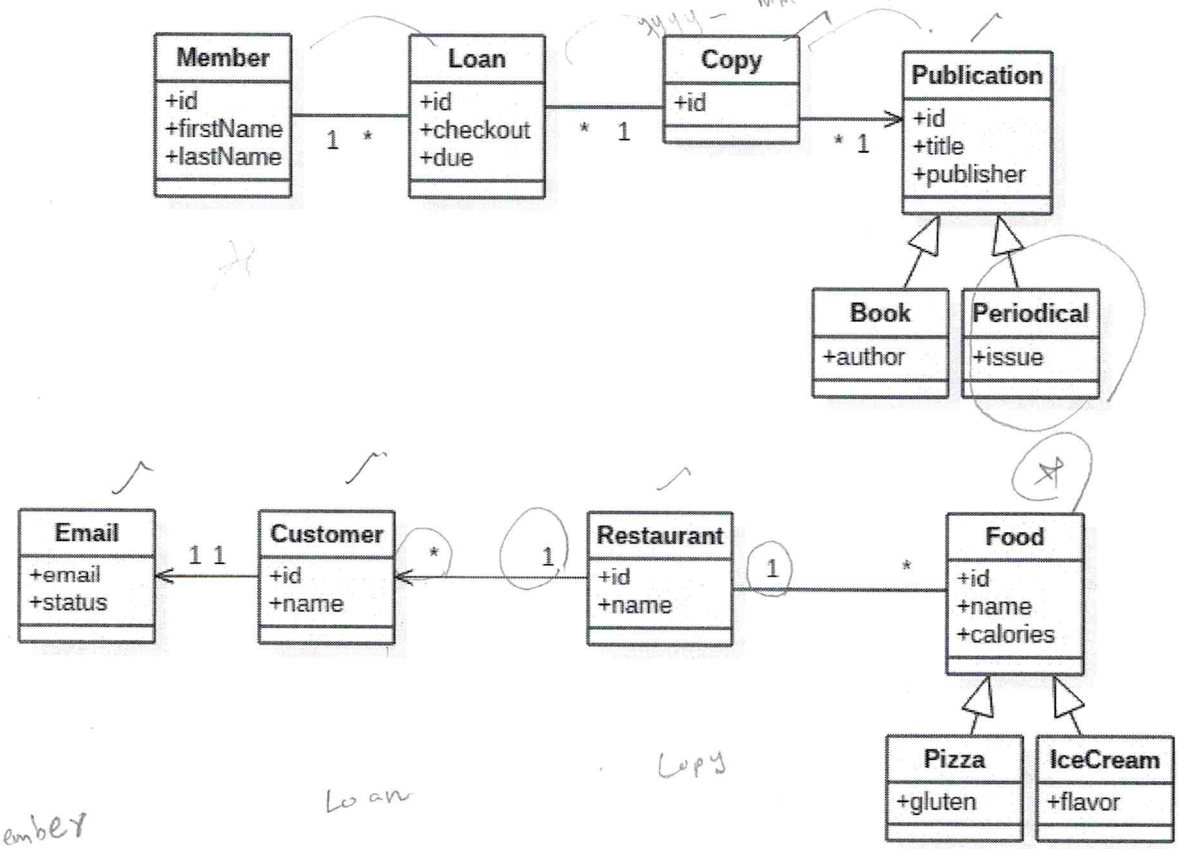
3 (XA) - Extended Architecture global transaction is a transactio that uses multiple transactional resources. It is managed by application side (using JTA). It uses 2 phase commit and is slow because of multiple resource connections. So, transactional resources become dependent on each other. It also needs to keep locks until all resources finished, again making things slower.

Name: Win Ei Khaing StudentID: 613403

These are UML diagrams of the domains used for the code exercises. I don't recommend using them for the mapping exercises (I may have forgotten to add or rename properties). They are meant for use with the JPQL queries to get an idea of how the different classes relate to each other.

The first domain is a Library domain, the second is a Restaurant domain

Hint about queries with dates: use the date directly in the string. For instance to get all loans that are due on 2022-01-23 write: from Loan l where l.due = '2022-01-23'



Name: Win Ei Khaing StudentID: 613403

Exercises:

1. [24 pts] Based on the following classes with annotations write what the tables names, column names, and data types will be (also include if a column is auto_increment).

```
@Entity
public class Member {
    @Id
    @GeneratedValue
    private Integer id;
    @Column(name="given")
    private String firstName;
    @Column(name="family")
    private String lastName;
    @OneToMany(mappedBy="member")
    private List<Loan> loans
        = new ArrayList<>();
}
@Entity
public class Loan {
    @Id
    @GeneratedValue
    private Long id;
    @ManyToOne
    private Member member;
    @ManyToOne
    private Copy copy;
    @Temporal(TemporalType.DATE)
    private Date checkout;
    @Temporal(TemporalType.DATE)
    private Date due;
    @Temporal(TemporalType.DATE)
    private Date returned;
}
@Entity
public class Copy {
    @Id
    @GeneratedValue
    private Long id;
    @OneToMany(mappedBy = "copy")
    private List<Loan> loans
        = new ArrayList<>();
    @ManyToOne
    private Publication publication;
}
```

```
@Entity
@Inheritance(strategy =
        InheritanceType.JOINED)
public abstract class Publication {
    @Id
    @GeneratedValue
    private Long id;
    private String title;
    private String publisher;
    @Lob
    private String text;
}
@Entity
public class Book extends Publication {
    private String author;
}
@Entity(name = "Magazine")
public class Periodical extends Publication {
    private String issue;
}
```

Member

id	int(11)	auto_increment
given	varchar(255)	
Family	varchar(255)	

Loan

id	bigint(20)	auto_increment
member_id	int(11)	
copy_id	bigint(20)	
checkout	date	
due	date	
returned	date	

Copy

id	bigint(20)	auto_increment
publication_id	bigint(20)	

Publication

id	bigint(20)	auto_increment
title	varchar(255)	
publisher	varchar(255)	
text	longtext	

Book

id	bigint(20)
author	varchar(255)

Magazine

id	bigint(20)
issue	varchar(255)

24

Name: Win Ei Khaing StudentID: 613403

2. [24 pts] Add annotations to the following classes to map to the tables shown on the next page.

```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @Embedded
    private Email mail;
}

@Embeddable
public class Email {

    private String email;

    private String status;
}

@Entity
public class Restaurant {
    @Id
    @GeneratedValue
    private Integer id;

    private String name;

    @OneToMany   ← default Jointable

    private List<Customer> customers =
        new ArrayList<>();

    @OneToMany(mappedBy = "restaurant")
    private List<Food> foods =
        new ArrayList<>();
}
```

```
@Entity
@Inheritance(strategy = InheritanceType.
                        TABLE_PER_CLASS)
public abstract class Food {
    @Id
    @GeneratedValue(strategy =
    private Long id; GenerationType.TABLE)

    private String name;

    @Column(name = "cals")
    private int calories;
    @ManyToOne
    @JoinColumn(name = "diner_id")
    private Restaurant restaurant;
}

@Entity
public class Pizza extends Food {

    private boolean gluten;
}

@Entity
public class IceCream extends Food {

    private String flavor;
}
```


24

Name: Win Ei Khaing StudentID: 613403

```
describe Customer;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| email  | varchar(255) | YES  |     | NULL    |                |
| status | varchar(255) | YES  |     | NULL    |                |
| name   | varchar(255) | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
describe Restaurant_Customer;
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| Restaurant_id | int(11)    | NO   | MUL | NULL    |       |
| customers_id  | bigint(20) | NO   | PRI | NULL    |       |
+---------------+------------+------+-----+---------+-------+
describe Restaurant;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255) | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
describe hibernate_sequences;
+------------------------+--------------+------+-----+---------+-------+
| Field                  | Type         | Null | Key | Default | Extra |
+------------------------+--------------+------+-----+---------+-------+
| sequence_name          | varchar(255) | NO   | PRI | NULL    |       |
| sequence_next_hi_value | bigint(20)   | YES  |     | NULL    |       |
+------------------------+--------------+------+-----+---------+-------+
describe Pizza;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| id       | bigint(20)   | NO   | PRI | NULL    |       |
| cals     | int(11)      | YES  |     | NULL    |       |
| name     | varchar(255) | YES  |     | NULL    |       |
| diner_id | int(11)      | YES  | MUL | NULL    |       |
| gluten   | bit(1)       | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
describe IceCream;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| id       | bigint(20)   | NO   | PRI | NULL    |       |
| cals     | int(11)      | YES  |     | NULL    |       |
| name     | varchar(255) | YES  |     | NULL    |       |
| diner_id | int(11)      | YES  | MUL | NULL    |       |
| flavor   | varchar(255) | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

Name: Win Ei Khaing StudentID: 613403

3. [12 pts] Based on the library domain write the following queries.
   a. All members who have a loan that is due on the 23rd of January 2022

4
```
SELECT DISTINCT m
FROM Member m JOIN m.loans l
WHERE l.due = '2022-01-23'
```

   b. All copies of the book with title "Dune"

4
```
FROM Copy c
WHERE c.publication.title LIKE 'Dune'
```

   c. All members who checked out the periodical titled "Communications of the ACM"

4
```
SELECT DISTINCT m
FROM Member m JOIN m.loans l
WHERE l.copy.publication.title LIKE 'Communications of the ACM'
AND type(l.copy.publication) = Magazine
```

4. [12 pts] Based on the restaurant domain write the following queries.
   a. All Customers whose email address ends in 'gmail.com'

4
```
FROM Customer c
WHERE c.mail.email LIKE '%gmail.com'
```

   b. All Customers who visited the restaurant "India Cafe"

4
```
SELECT DISTINCT c
FROM Restaurant r JOIN r.customers c
WHERE r.name LIKE 'India Cafe'
```

   c. All Customers who ate the pizza with name'Californian' at the restaurant 'Revelations'

4
```
SELECT DISTINCT c
FROM Restaurant r JOIN r.customers c
WHERE r.name LIKE 'Revelations' AND
r.id IN ( SELECT res.id
          FROM Food f JOIN f.restaurant res
          WHERE f.name LIKE 'Californian' AND
          type(f) = Pizza )
```


5 alternative →

```
SELECT  DISTINCT  r.customers
FROM    Food f    JOIN    f.restaurant  r
WHERE   f.name    LIKE  'Californian'  AND
        type(f)   =     Pizza          AND
        r.name    LIKE  'Revelations'
```

this one is better, no sub-select