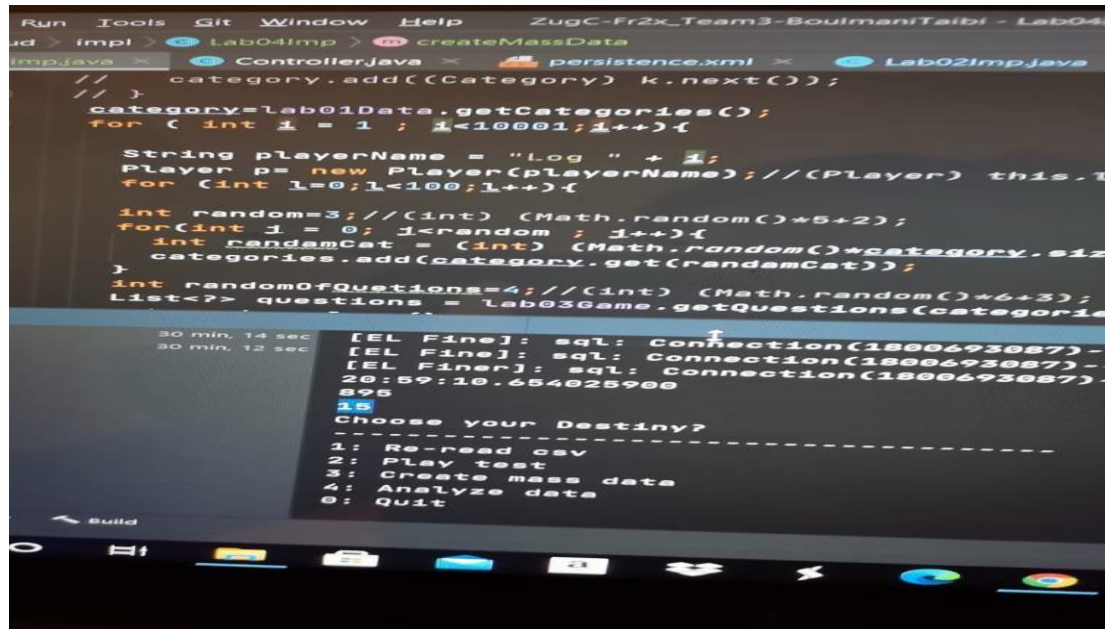


PraktikumsBericht

Die Massendatengenerierung bei der Anwendung hat bei uns 15 Minuten gedauert.



The screenshot shows the Eclipse IDE with several tabs open: 'Lab04Imp', 'Controller.java', 'persistence.xml', and 'Lab02Imp.java'. The 'Controller.java' tab is active, displaying Java code for mass data generation. The code includes a loop to create categories, a loop to create players, and a loop to create questions. The console output shows the execution of the application, including SQL connection logs and a menu prompt 'Choose your Destiny?'. The menu options are: 1: Re-read csv, 2: Play test, 3: Create mass data, 4: Analyze data, 0: Quit. The user has selected option 3, 'Create mass data'.

Wir haben erstmal in persistence.xml zwei properties hinzugefügt.

```
<property name="eclipselink.jdbc.batch-writing" value="JDBC"/>
```

Um den zu verwendenden Batch-Schreibtyp zu konfigurieren, haben wir hier JDBC batch writing verwendet, indem beim Batch-Writing EclipseLink Transaktionen mit mehreren Schreibfunktionen optimieren kann.

```
<property name="eclipselink.jdbc.batch-writing.size" value="1000"/>
```

Bei parametrisiertem Batch-Schreiben ist dieser Wert die Anzahl der Anweisungen zum Batch.

```

public class Category {
    @Id
    @GeneratedValue( strategy = GenerationType.AUTO, generator = "public.id_category" )
    @SequenceGenerator( name="public.id_category", sequenceName="public.id_category", initialValue=1, allocationSize=1000)
    private int id;
    @Column(name = "name", unique = true)
    private String name;
    @OneToMany (fetch = FetchType.EAGER )//(mappedBy = "category", cascade = {CascadeType.PERSIST})
    @JoinColumn(name = "Category_id", nullable = false)
    private List<Question> myQuestions;
}

```

Wir haben ein fetch in Type EAGER in Beziehung @OneToMany List<Question> in Klasse Category hinzugefügt, d.h wollten wir mal hier alle Questions für die Category sofort vollständig geladen werden, Um es zusammen mit den restlichen Feldern zu laden (d.h. eifrig)

```

public class Player {
    @Id
    private String name;
    @OneToMany (mappedBy = "player", cascade = CascadeType.PERSIST)
    private List<Game> games;
}

```

(cascade = CascadeType.PERSIST) würde in @OneToMany List<Game> in der Entity Player Hinzufügen

Das heißt ein expliziter Aufruf von persist(Game) ist wegen CascadeType.PERSIST nicht mehr nötig.

Der CascadeType.PERSIST gibt den Persistenzvorgang von einer übergeordneten an eine untergeordnete Entität weiter.

```

24 List resultL = lab01Data.getCategories();
25 for(int i = 1; i <= 10000; i++){
26     // Player p = (Player) lab03Game.getOrCreatePlayer("Player"+c);
27     Player p = new Player( name: "Player"+c);
28     for(int j = 1; j <= 100; j++){
29         int random, count = 0;
30         List categories = new ArrayList();
31         int countcategories = rand.nextInt( bound: (5-3)+1)+3;
32         while (count < countcategories){
33             random = rand.nextInt( bound: ((resultL.size()-1)-0)+1)+0;;
34             categories.add(resultL.get(random));
35             count++;
36         }
37         int amountOfQuestionForCategory = rand.nextInt( bound: (5-3)+1)+3;
38         Game game = (Game) lab03Game.createGame(p, lab03Game.getQuestions(categories, amountOfQuestionForCategory));
39         lab03Game.playGame(game);
40         p.getGames().add(game);
41         categories.clear();
42         // lab03Game.persistGame(game);
43     }
44     em.persist(p);
45     if(i % 1000 == 0){
46         em.flush();
47         em.clear();
48     }

```

Wenn Sie die Zeile 44 schauen: Nachdem ein Spieler 100 Spiele spielt, nimmt Persist eine Entitätsinstanz Player, fügt sie dem Kontext hinzu und macht diese Instanz verwaltet, und wir brauchen wie oben erklärt würde kein expliziter Aufruf von Persist Game, d.h für jeden Persist vom 1 Player wird 100 Games auch Persist.

```

44     em.persist(p);
45     if(i % 1000 == 0){
46         em.flush();
47         em.clear();
48     }
49     c++;
50 }
51 em.getTransaction().commit();
52 em.close();

```

In Zeile 45: wenn $i \bmod 1000 == 0$, bzw. Wenn 1000 Players persist sind, rufen wir **flush()** Methode in Zeile 46 auf, indem hier flush() verwendet werden kann, um alle Änderungen

in die Datenbank zu schreiben, bevor die Transaktion bestätigt wird.

Nachdem `flush()` Methode ausgeführt wird, wird dann die `clear()` Methode aufgerufen werden, indem durch das Löschen des Entity-Managers der zugehörige Cache geleert wird, wodurch neue Datenbankabfragen zu einem späteren Zeitpunkt der Transaktion ausgeführt werden können.

Queries um zu nachweisen, die korrekte Anzahl an Datensätzen generiert wurde.

```
postgres=# select count(*) from player;
count
-----
10000
(1 row)

postgres=# select count(*) from game;
count
-----
990900
(1 row)

postgres=# select count(*) from game_myquestionmap;
count
-----
9351681
(1 row)
```