PROBLÉMA-ANALÍZIS ENTERPRISE RENDSZEREKBEN

7. forduló

adnovum

A kategória támogatója: Adnovum Hungary Kft.

RENDELKEZÉSRE ÁLLÓ IDŐ:

15:00

Ismertető a feladathoz

Csapatunk egy dinamikusan fejlődő startup, termékünk nagyon sikeres így folyamatosan növekszik a felasználóink száma. Emellett új komponensek és régiók kerülnek bevezetésre. Szükségünk van egy rátermett fejlesztőre, aki képes a felmerülő tervezési, biztonsági és teljesítmény problémákat megoldani. Elvállalod?

Felhasznált idő: 01:48/15:00 **Elért pontszám:** 0/27

1. feladat 0/7 pont

A következők közül melyek okoznak N+1 problémát?

Válaszok

- [ ] userIds = select id from user; select * from setting where user_id in (userIds);
- [x] userIds = select id from user;
  for each (userId in userIds) {
  select * from setting where user_id = userId;
- [x] orderIds = select id from order;
  for each (orderId in orderIds) {
  select * from order_item where order_id = orderId;

```
}
```

- [x] userIds = select distinct(user_id) from setting;

```
for each (userId in userIds) {
 select * from user where id = userId;
```

Magyarázat

N darab lekérést végzünk el a kezdeti lista lekérés mellé: N+1

2. feladat 0/12 pont

A következő két osztály egy Spring Boot-ra épülő alkalmazásban található, ami a Spring Data komponenst használja, hogy az általa használt adatokat adatbázisba perzisztálja.

```
@RestController
@Transactional
public class UserController {
    @Autowired private UserService userService;

    @PostMapping("/update")
    public void updateUser(Long userId, String firstName, String lastName, String avatar) {
        userService.updateProfile(userId, firstName, lastName);
        userService.setAvatar(userId, avatar);
    }
}

@Service
@Transactional(propagation = Propagation.REQUIRES_NEW)
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public void updateProfile(Long userId, String firstName, String lastName) {
        User user = userRepository.findById(userId).get();
        user.setFirstName(firstName);
        user.setLastName(lastName);
        try {
            calculateTimestamp(userId);
        }
        catch (Exception e) {
            //nothing to do
        }
        userRepository.save(user);
```

```
    }

    @Transactional(propagation = Propagation.REQUIRES_NEW)
    private void calculateTimestamp(Long userId) {
        User user = userRepository.findById(userId).get();

        LocalDateTime ldt = LocalDateTime.now();
        String dateStamp = DateTimeFormatter.ofPattern("yyyy-MM-dd", Locale.ENGLISH).format(ldt);
        user.setLastUpdated(dateStamp);

        // Boom!
        throw new SimulationException("Oh-oh!");
    }

    public void setAvatar(Long userId, String avatar) {
        if (avatar == null || avatar.isEmpty()) {
            throw new IllegalArgumentException("avatar is not specified");
        }
        userRepository.findById(userId).get().setAvatar(avatar);
    }
}
```

A rendszer dependency injectionhoz a Spring alapértelmezett AOP működésére támaszkodva proxy objektumokkal dolgozik.

Tegyük fel, hogy a rendszerünkben van egy felhasználó az alábbi tulajdonságokal:

id=1

firstName=John

lastName=Doe

avatar=<egy kép base64 kódolásban>

lastUpdated=2021-03-01

A rendszer egyik adminisztrátora a következő adatokkal hívja meg az UserController-ben definiált végpontot:

id=1

firstName=Jane

lastName=Doe

avatar=null

Az alábbi állítások közül mely lesz igaz az 1-es azonosítójú felhasználóra a művelet után?

Válaszok

- [ ] firstName=John
- [x] firstName=Jane
- [ ] firstName=null
- [x] lastName=Doe
- [ ] lastName=null
- [x] avatar=<egy kép base64 kódolásban>

- [ ] avatar=null
- [x] lastUpdated=<aktuális dátum yyyy-MM-dd formátumban>
- [ ] lastUpdated=2021-03-01
- [ ] lastUpdated=null

Magyarázat

Az updateProfile és a setAvatar külön tranzakcióban fut. Az utóbbi meghiúsul, hiszen paraméternek null-t adtunk át, így az avatar értéke nem változik.

Ettől függetlenül az updateProfile elvégzi firstName és a lastName módosítását hiszen a különálló tranzakcióját nem hiúsítja meg az avatar módosítás sikertelensége.

A lastUpdated metódus noha meg van jelölve, hogy külön tranzakcióban fusson (@Transactional(propagation = Propagation.REQUIRES_NEW)), az annotáció nem érvényesül, mert a hívás az objektumon belülről érkezik, nem pedig kintről, a proxy-n keresztül.

3. feladat 0/8 pont

A tervezett szolgáltatással kapcsolatban fontos szempont, hogy ellenálló legyen Denial of Service támadás ellen. Milyen módszerekkel csökkenthetjük az esélyét, hogy egy ilyen támadás szolgáltatás kiesést okozzon?

Válaszok

- [x] statikus fájlok kiszolgálása Content Delivery Network segítségével
- [ ] minden hálózati kommunikációt HTTPS protokollon valósítunk meg
- [x] olyan architektúrát alkalmazunk, ami a terhelés függvényében képes dinamikusan skálázódni
- [x] kártékony forgalom szűrése a tűzfalnál
- [x] válaszok cachelése
- [x] rate-limiting
- [x] aszinkron kommunikáció
- [x] circuit breaker pattern használata

Magyarázat

1. válasz: Igen. A CDN célja statikus fájlok hatékony kiszolgálása, alkalmazása jelentős terheléstől mentesítheti a szolgáltatásunkat.

2. válasz: Nem. A HTTPS, bár biztonsági szempontból előnyös, alkalmazása nem segít ebben a helyzetben.
3. válasz: Igen. Ha megállítani nem is sikerül teljesen terheléses támadást, ha az alkalmazásunk el tudja nyelni a fennmaradó részét azzal megelőzhető a szolgáltatáskiesés.
4. válasz: Igen, például ha nem elosztott támadásról van szó, amely egy IP címről érkezik.
5. válasz: cachelt válaszok kevesebb rendszertelhelést okoznak
6. válasz: IP cim szerint limitálni a forgalmat
7. válasz: Üzenetek késleltetett feldolgozása lehetőséget ad a rendszernek a kapacitásához mérten feldolgozni az üzeneteket
8. válasz: A circuit breaker pattern segitségével elkerülhetőek azok a hivások amik amugy is sikertelenek lennének, ezzel csökkentve a terhelést