**Opdracht**

De code uit het bestand bank.zip vormt de basis van de volgende opdrachten. Elke subopdracht bestaat uit een (gedeelte van) testscipt. Integreer de testscipts in de bank applicatie. Zorg ervoor dat voor elk testscript de database leeg gemaakt wordt. Je kunt hiervoor het script DatabaseCleaner gebruiken.

Voor elke test moet je in ieder geval de volgende vragen beantwoorden:

1. Wat is de waarde van asserties en printstatements? Corrigeer verkeerde asserties zodat de test ‘groen’ wordt.
2. Welke SQL statements worden gegenereerd?
3. Wat is het eindresultaat in de database?
4. Verklaring van bovenstaande drie observaties.

De antwoorden op de vragen kun je als commentaar bij de testen vastleggen.

1 **Hoe werken persist en commit in samenhang met de database.**

Account account = new Account(111L);

em.getTransaction().begin();

em.persist(account);

//TODO: verklaar en pas eventueel aan

assertNull(account.getId());

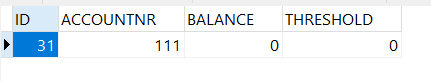
em.getTransaction().commit();

System.out.println(“AccountId: “ + account.getId());

//TODO: verklaar en pas eventueel aan

assertTrue(account.getId() > 0L);

**Antwoorden**

1. AccountId: 5
2. “SELECT a.Id FROM Account”
3. 
4. Account werd niet gezien als entity omdat hij nog niet in de presistant.xml staat. Het ID bestond al, de database werd nog niet gecleaned, kijk naar de @before, hier staat de databasecleaner.

2 **Rollback**

Account account = new Account(111L);

em.getTransaction().begin();

em.persist(account);

assertNull(account.getId());

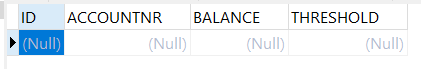
em.getTransaction().rollback();

// TODO code om te testen dat table account geen records bevat. Hint: bestudeer/gebruik AccountDAOJPAImpl

**Antwoorden**

1. *assertEquals*(0, new AccountDAOJPAImpl(em).count());  
   *assertTrue*(new AccountDAOJPAImpl(em).findAll().isEmpty());
2. INSERT INTO Account (ID, ACCOUNTNR)  
   VALUES (1, 111L);

ROLLBACK…

1. 
2. We moesten kijken welke functie in AccountDAOJPAImpl alles ophaalde uit de database en daarmee moesten we aan kijken of deze daadwerkelijk leeg was.

3 **Flushen maar**

Long expected = -100L;

Account account = new Account(111L);

account.setId(expected);

em.getTransaction().begin();

em.persist(account);

//TODO: verklaar en pas eventueel aan

//assertNotEquals(expected, account.getId();

em.flush();

//TODO: verklaar en pas eventueel aan

//assertEquals(expected, account.getId();

em.getTransaction().commit();

//TODO: verklaar en pas eventueel aan

**Antwoorden**

1. De onderstaande twee lijnen hebben we veranderd.

assertNotEquals(expected, account.getId();

em.flush();

Deze waren incorrect en hebben we aangepast naar:

em.flush();

assertNotEquals(expected, account.getId());

1. INSERT INTO Account (ID, ACCOUNTNR)  
   VALUES (1, 111L);
2. 
3. In de code word er het ID aangepast, dit mag echter niet wegens dat hij auto generated is. De flush zorg ervoor dat de overige data die nog niet verstuurd is, verstuurd word.

4 **Veranderingen na de persist**

Long expectedBalance = 400L;

Account account = new Account(114L);

em.getTransaction().begin();

em.persist(account);

account.setBalance(expectedBalance);

em.getTransaction().commit();

assertEquals(expectedBalance, account.getBalance());

//TODO: verklaar de waarde van account.getBalance

Long cid = account.getId();

account = null;

EntityManager em2 = emf.createEntityManager();

em2.getTransaction().begin();

Account found = em2.find(Account.class, cid);

//TODO: verklaar de waarde van found.getBalance

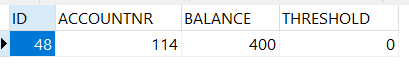
assertEquals(expectedBalance, found.getBalance());

**Antwoorden**

1. Added a em2.close();
2. INSERT INTO ACCOUNT (ACCOUNTNR, BALANCE, THRESHOLD) VALUES (?, ?, ?) bind => [114, 400, 0]

SELECT LAST\_INSERT\_ID()

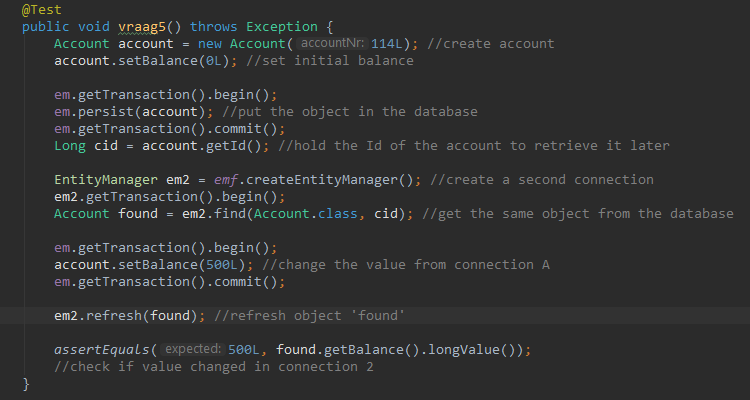
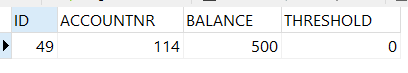
SELECT ID, ACCOUNTNR, BALANCE, THRESHOLD FROM ACCOUNT WHERE (ID = ?) bind => [4]

1. 
2. Door de transaction te gebruiken worden alle veranderingen meegenomen. Vanaf het begin tot aan de commit.

5 **Refresh**

In de vorige opdracht verwijzen de objecten account en found naar dezelfde rij in de database. Pas een van de objecten aan, persisteer naar de database. Refresh vervolgens het andere object om de veranderde state uit de database te halen. Test met asserties dat dit gelukt is.

**Antwoorden**

1. 
2. Account = SELECT \* FROM Account a WHERE a.Id = Account.getId()
3. 
4. Het object word refreshed met wat er op datmoment in de database staat.

6 **Merge**

Merge is een van de lastigere methoden uit JPA api. Het is belangrijk dat je deze opgave daarom zorgvuldig uitvoert.

Account acc = new Account(1L);

Account acc2 = new Account(2L);

Account acc9 = new Account(9L);

// scenario 1

Long balance1 = 100L;

em.getTransaction().begin();

em.persist(acc);

acc.setBalance(balance1);

em.getTransaction().commit();

//TODO: voeg asserties toe om je verwachte waarde van de attributen te verifieren.

//TODO: doe dit zowel voor de bovenstaande java objecten als voor opnieuw bij de entitymanager opgevraagde objecten met overeenkomstig Id.

**Antwoorden**

1. Iets
2. Iets
3. Iets
4. iets

// scenario 2

Long balance2a = 211L;

acc = new Account(2L);

em.getTransaction().begin();

acc9 = em.merge(acc);

acc.setBalance(balance2a);

acc9.setBalance(balance2a+balance2a);

em.getTransaction().commit();

//TODO: voeg asserties toe om je verwachte waarde van de attributen te verifiëren.

//TODO: doe dit zowel voor de bovenstaande java objecten als voor opnieuw bij de entitymanager opgevraagde objecten met overeenkomstig Id.

// HINT: gebruik acccountDAO.findByAccountNr

**Antwoorden**

1. Iets
2. Iets
3. Iets
4. iets

// scenario 3

Long balance3b = 322L;

Long balance3c = 333L;

acc = new Account(3L);

em.getTransaction().begin();

Account acc2 = em.merge(acc);

assertTrue(em.contains(acc)); // verklaar

assertTrue(em.contains(acc2)); // verklaar

assertEqual(acc,acc2); //verklaar

acc2.setBalance(balance3b);

acc.setBalance(balance3c);

em.getTransaction().commit() ;

//TODO: voeg asserties toe om je verwachte waarde van de attributen te verifiëren.

//TODO: doe dit zowel voor de bovenstaande java objecten als voor opnieuw bij de entitymanager opgevraagde objecten met overeenkomstig Id.

**Antwoorden**

1. Iets
2. Iets
3. Iets
4. iets

// scenario 4

Account account = new Account(114L) ;

account.setBalance(450L) ;

EntityManager em = emf.createEntityManager() ;

em.getTransaction().begin() ;

em.persist(account) ;

em.getTransaction().commit() ;

Account account2 = new Account(114L) ;

Account tweedeAccountObject = account2 ;

tweedeAccountObject.setBalance(650l) ;

assertEquals((Long)650L,account2.getBalance()) ; //verklaar

account2.setId(account.getId()) ;

em.getTransaction().begin() ;

account2 = em.merge(account2) ;

assertSame(account,account2) ; //verklaar

assertTrue(em.contains(account2)) ; //verklaar

assertFalse(em.contains(tweedeAccountObject)) ; //verklaar

tweedeAccountObject.setBalance(850l) ;

assertEquals((Long)650L,account.getBalance()) ; //verklaar

assertEquals((Long)650L,account2.getBalance()) ; //verklaar

em.getTransaction().commit() ;

em.close() ;

**Antwoorden**

1. Iets
2. Iets
3. Iets
4. iets

7 **Find en clear**

Account acc1 = new Account(77L);

em.getTransaction().begin();

em.persist(acc1);

em.getTransaction().commit();

//Database bevat nu een account.

// scenario 1

Account accF1;

Account accF2;

accF1 = em.find(Account.class, acc1.getId());

accF2 = em.find(Account.class, acc1.getId());

assertSame(accF1, accF2);

// scenario 2

accF1 = em.find(Account.class, acc1.getId());

em.clear();

accF2 = em.find(Account.class, acc1.getId());

assertSame(accF1, accF2);

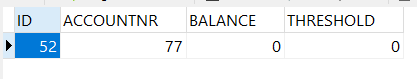
//TODO verklaar verschil tussen beide scenario’s

**Antwoorden**

1. -
2. INSERT INTO ACCOUNT (ACCOUNTNR, BALANCE, THRESHOLD) VALUES (?, ?, ?) bind => [77, 0, 0]

SELECT LAST\_INSERT\_ID()

SELECT ID, ACCOUNTNR, BALANCE, THRESHOLD FROM ACCOUNT WHERE (ID = ?) bind => [3]

1. 
2. In scenario 1 zijn beide objecten uit de database gehaald waarbij het zelfde object wordt teruggestuurd maar op twee verschillende punten in het geheugen (pointers worden vergeleken '=='). In scenario 2 wordt nadat accF1 is opgehaald de context van de EntityManager gecleared waarna het tweede object wordt opgehaald

8 **Remove**

Account acc1 = new Account(88L);

em.getTransaction().begin();

em.persist(acc1);

em.getTransaction().commit();

Long id = acc1.getId();

//Database bevat nu een account.

Em.remove(acc1);

assertEquals(id, acc1.getId());

Account accFound = em.find(Account.class, id);

assertNull(accFound);

//TODO: verklaar bovenstaande asserts

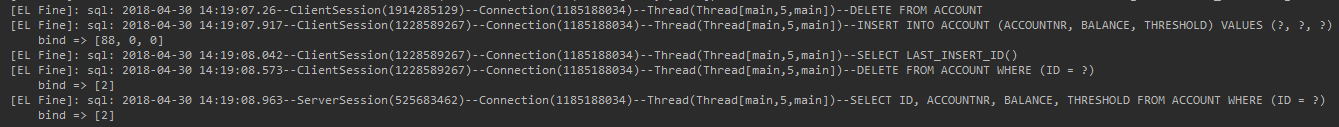
**Antwoorden**

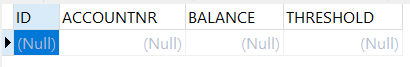
1. added em.commit() want anders commit hij hem niet en staat het gemaakte account nog steeds in de database.

em.getTransaction().begin();

em.remove(acc1); //changed "Em.remove(acc1)" to current

em.getTransaction().commit();

2. 

3. 

4. maakt hem aan en gooit hem weg

9 **Generation type**

Opgave 1 heb je uitgevoerd met @GeneratedValue(strategy = GenerationType.IDENTITY)

Voer dezelfde opdracht nu uit met GenerationType SEQUENCE en TABLE.

Verklaar zowel de verschillen in testresultaat als verschillen van de database structuur.

@GeneratedValue(strategy = GenerationType. SEQUENCE)

Sequence word op een andere tijd aangeroepen dan identity. Identity bij een commit, sequence al bij persistence.

Vraag 1. Alleen het id is veranderd.

@GeneratedValue(strategy = GenerationType. TABLE)

Vraag 1.

Hij maakt een apart table aan waarbij hij een sequence bijhoud

Helaas weten we niet waarom de waardes die we zien zo anders van elkaar zijn.

