HAAGA-HELIA
ammattikorkeakoulu

Samanaikaisuuden hallinta MySQL-tietokannanhallintajärjestelmässä käyttäen InnoDB-tietokantamoottoria

Vesa Tähkävuori

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2010

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

16.11.2010

Tietojenkäsittelyn koulutusohjelma, Pasilan toimipiste

Tekijät Vesa Tähkävuori	**Ryhmä tai aloitusvuosi** 2004
Opinnäytetyön nimi Samanaikaisuuden hallinta MySQL-tietokannanhallintajärjestelmässä käyttäen InnoDB-tietokantamoottoria	**Sivu- ja liitesivumäärä** 42 + 130
Ohjaaja tai ohjaajat Martti Laiho	

Vuonna 1995 ensimmäistä kertaa julkaistu MySQL on kehittynyt nopeasti yhdeksi maailman suosituimmista tietokannanhallintajärjestelmistä (TKHJ), joka kykenee kilpailemaan kolmen yleisimmän TKHJ:n: Oraclen, DB2:n ja SQL Serverin kanssa usealla osa-alueella. Se on saavuttanut suosiota erityisesti osana verkkosovelluksia.

MySQL:n arkkitehtuurissa tietokannan ydin on erotettu tauluja käsittelevästä osasta. Jälkimmäistä kutsutaan tietokantamoottoriksi, jonka valinta vaikuttaa ratkaisevasti MySQL:n käyttäytymiseen. Yksi MySQL:n yleisimmistä tietokantamoottoreista on InnoDB, joka on uusimmissa versioissa valittu myös oletustietokantamoottoriksi.

Yksi keskeisimmistä TKHJ:n (MySQL:n tapauksessa tietokantamoottorin) ratkaistavista ongelmista liittyy tilanteisiin, joissa tietokannassa olevaa dataa yritetään päivittää (ja mahdollisesti lukea) samanaikaisesti useasta eri lähteestä. Tämän tutkimuksen tarkoituksena oli selvittää, miten tämä ns. samanaikaisuuden hallinta on toteutettu InnoDB:ssä. Tutkimuksessa keskityttiin erityisesti siihen, miten InnoDB selviää ennalta määritellyistä samanaikaisuuden hallinnan ongelmaskenaarioista.

Tutkimuksessa todettiin, että tiukimmilla asetuksilla InnoDB selviää samanaikaisuuden hallinnasta erinomaisesti. Löyhemmillä asetuksilla esiintyneet ongelmat olivat pääsääntöisesti SQL-standardin mukaisia. Lisäksi tutkimuksessa selvisi varotoimenpiteitä, joita MySQL/InnoDB -yhdistelmän kanssa työskentelevien sovelluskehittäjien kannattaa ottaa huomioon.

Asiasanat
SQL, standardit, tietokannat, samanaikaisuuden hallinta

HAAGA-HELIA
ammattikorkeakoulu

Abstract

November 16, 2010.

Degree Programme in Business Information Technology, Pasila Campus

Authors Vesa Tähkävuori	**Group or year of entry** 2004
The title of thesis Concurrency control in MySQL DBMS using the InnoDB storage engine	**Number of pages and appendices** 42 + 130
Supervisors Martti Laiho	

MySQL, initially released in 1995, has rapidly become one of the most popular database management systems (DBMS) in the world. It is capable of competing with the three most common DBMSs: Oracle, DB2 and SQL Server in several fields. It has become particularly popular as a part of web applications.

In MySQL's software architecture, the database core is separated from the part which handles the tables in the database. The latter is called a storage engine, the choice of which makes a crucial difference in MySQL's behavior. One of the most common storage engines for MySQL is InnoDB, which has also been chosen as the default storage engine for latest versions.

One of the most essential problems that the DBMS (or in MySQL's case the storage engine) has to solve is related to situations where data located in the database is being updated (and possibly read) from multiple sources at the same time. The purpose of this thesis was to clarify how this so called concurrency control has been implemented in InnoDB. The primary focus of the study was on observing InnoDB's behavior in several predefined problematic concurrency control scenarios.

The study indicated that InnoDB's concurrency control was excellent when the tightest possible settings were used. The problems that appeared when using looser settings were mostly in accordance with the SQL standard. Furthermore, the study revealed some additional safety measures that software developers should take into consideration when working with the MySQL/InnoDB combination.

Key words
SQL, standards, databases, concurrency control

Sisällys

1 Johdanto

Tietokantojen merkitystä maailmalle ei voida korostaa liikaa. Nykyinen elämäntyyli on jo vuosikymmeniä ollut täysin riippuvainen kyvystä tallentaa ja lukea valtavia määriä tietoa sekä nopeasti että luotettavasti. Vaikka tietokannat ovat monella tapaa vakiintuneet vuosien saatossa, löytyy useisiin niitä koskeviin ongelmiin yhä useita kilpailevia ratkaisuja. Yksi keskeisimmistä ongelmista liittyy tilanteisiin, joissa tietokannassa olevaa tietoa yritetään lukea ja päivittää samanaikaisesti useasta eri lähteestä. Tässä yhteydessä puhutaan niin sanotusta samanaikaisuuden hallinnasta. Jopa yleisimpien tietokannanhallintajärjestelmien välillä löytyy merkittäviä eroja sen suhteen, miten samanaikaisuuden hallinta on ratkaistu. Lisäksi löytyy useita muitakin samanaikaisuuden hallinnan teknologioita, joista osa on huomattavasti harvinaisempia tai jäänyt pelkän teorian tasolle.

Alun perin vuonna 1995 julkaistu MySQL on kymmenien miljoonien käyttäjiensä avulla noussut nopeasti yhdeksi maailman yleisimmistä tietokannanhallintajärjestelmistä. MySQL kykenee kilpailemaan kolmen yleisimmän tietokannanhallintajärjestelmän: Oraclen, DB2:n ja SQL Serverin kanssa usealla osa-alueella. Nykyään voidaankin hyvällä syyllä puhua ”kolmen suuren” sijaan ”neljästä suuresta”. Erityisen suosittu MySQL on verkkopohjaisten sovellusten kehittämisessä. MySQL:n suosion kannalta oleellista on ollut se, että se pohjautuu vapaaseen lähdekoodiin ja sen täysi versio on ladattavissa ilmaiseksi valtaosaan projekteista.

MySQL eroaa monista tietokannanhallintajärjestelmistä myös siinä, että MySQL:ssä tietokannan ydin on erotettu tauluja käsittelevästä koodista. Tätä osaa kutsutaan tietokantamoottoriksi. MySQL:ään voi ladata useita kymmeniä eri tietokantamoottoreita jotka vaikuttavat ratkaisevasti sen käyttäytymiseen. MySQL:n oletustietokantamoottori on nykyään tässä opinnäytetyössä käsiteltävä InnoDB. Voidaan olettaa, että hyvin merkittävä osa MySQL:ää käyttävistä sovelluksista tulee tulevaisuudessakin käyttämään InnoDB:tä tietokantamoottorinaan.

1.1 Tavoitteet ja tutkimusongelmat

Tässä opinnäytetyössä tutkitaan InnoDB-tietokantamoottorin toteuttamaa samanaikaisuuden hallintaa. Työn tarkoituksena on selvittää, miten MySQL:n InnoDB-tietokantamoottori käyttäytyy ennalta määritellyissä samanaikaisuuden hallinnan skenaarioissa. Tämä on myös tutkimuksen pääongelma. Tarkemmin määriteltynä tarkoitus on selvittää, mikä on tietokannan lopputila skenaarion suorittamisen jälkeen, mitkä ovat MySQL:n antamat ilmoitukset ja mitä luk-

koja InnoDB on kunkin vaiheen suorittamisen jälkeen ottanut. Näiden tietojen avulla voidaan auttaa sovelluskehittäjiä välttämään samanaikaisuuden hallinnan ongelmatilanteita.

Lisäksi työssä vertaillaan InnoDB-tietokantamoottorin samanaikaisuuden hallintaa kolmeen yleisesti yritysmaailmassa käytössä olevan tietokannanhallintajärjestelmän: Oraclen, DB2:en ja SQL Serverin vastaavaan. Tämä vertailu voi olla arvokas yritykselle tai yhteisölle joka harkitsee vaihtoehtoja eri tietokantojen hallintajärjestelmien välillä.

Tutkimuksen aliongelmat voidaan esittää seuraavalla tavalla:

- Miten MySQL:n InnoDB-tietokantamoottori toteuttaa SQL-standardin mukaiset eristyvyystasot?
- Miten InnoDB:ssä toteutettu samanaikaisuuden hallinta eroaa Oraclen, DB2:n ja SQL Serverin vastaavista?

1.2 Tutkimuksen rajaus

Tämän tutkimuksen tulokset on rajattu koskemaan MySQL Community Serverin uusinta vakaata julkaisua 5.1.49.

Se, miten MySQL tukee samanaikaisuuden hallintaa, riippuu käytettävästä tietokantamoottorista. Työ on rajattu koskemaan InnoDB-tietokantamoottorin uusinta vakaata versiota 1.0.12.

Samanaikaisuuden hallinta viittaa tässä työssä erityisesti seuraaviin ongelmiin ja niihin liittyviin ratkaisuihin:

- Hukatun päivityksen ongelma (Lost Update Problem)
- Likainen luku (Dirty Read Problem)
- Muuttuva lukujoukko (Non-Repeatable Read)
- Kasvava lukujoukko (Phantom Problem)
- Ristiinpäivittävä UPDATE-UPDATE -kilpailu
- Isärivin poistaminen (Delete Parent Problem)
- Rakenteen muuttaminen

Yllämainitut ongelmat käsitellään teoreettisessa taustassa alkaen luvusta **Samanaikaisuuden hallinta**.

Tutkimuksessa tutkitaan ainoastaan kahta samanaikaista samalla eristyvyystasolla toimivaa transaktiota. Useamman kuin kahden tai eri eristyvyystasoja käyttävien transaktioiden tutkiminen ei mahdu tämän opinnäytetyön laajuuteen ja on siten rajattu tarkastelun ulkopuolelle.

Vertailu Oracleen, DB2:een ja SQL Serveriin koskee seuraavia versioita:

- Oracle 11g1
- DB2 Express-C 9.7
- SQL Server 2008

Vertailussa keskitytään vain siihen, miten kyseisten järjestelmien samanaikaisuuden hallinta on toteutettu. Suorituskyky rajataan tarkastelun ulkopuolelle.

SQL-standardilla tarkoitetaan tässä uusinta ANSI SQL:2008-standardia, joskin valtaosa tässä opinnäytetyössä käsiteltävistä asioista on peräisin SQL:n vanhemmista standardeista ja on säilynyt nykypäivään asti muuttumattomana.

2 MySQL-tietokannanhallintajärjestelmästä

2.1 MySQL

MySQL on MySQL AB:n kehittämä suosittu avoimen lähdekoodin tietokannanhallintajärjestelmä (Wikipedia 2010a). MySQL AB:n perustivat suomalainen Michael Widenius ja ruotsalaiset David Axmark ja Allan Larsson vuonna 1995. Vuonna 2008 yhdysvaltalainen Sun Microsystems osti yrityksen. Sun Microsystems on vuodesta 2009 lähtien ollut tietokannanhallintajärjestelmiä kehittävän Oraclen tytäryhtiö. (Wikipedia 2010b, MySQL 2008, Oracle 2009.) MySQL on nimetty Wideniuksen My-tyttären mukaan (MySQL 2010a).

MySQL on ladattavissa ilmaiseksi GPLv2-lisenssillä avoimen lähdekoodin projekteihin. Yritysasiakkaat voivat ladata maksullisen MySQL Enterprise-tuotteen, joka mahdollistaa MySQL:n integroimisen osaksi suljetun lähdekoodin järjestelmiä. Tuotteeseen sisältyy ympärivuorokautinen tuki. MySQL on saatavissa usealle eri käyttöjärjestelmälle, mm. Windowsille, Linuxille ja Mac OS X:lle. (MySQL 2010b; Wikipedia 2010a.)

2.2 MySQL:n eri tietokantamoottorit (Storage engines)

MySQL:n arkkitehtuurissa tietokannan ydin on erotettu koodista, joka käsittelee tauluja. Tällainen arkkitehtuuri on tietokannanhallintajärjestelmälle harvinainen, joskaan ei täysin ainutlaatuinen. Koska tietokantamoottori määritellään jokaiselle taululle erikseen, sitä on joskus kutsuttu myös nimellä taulun tyyppi (table type). MySQL:ään on tarjolla kymmeniä yleisessä tiedossa olevia tietokantamoottoreita, joista suurin osa on ulkopuolisen yrityksen kehittämiä. Suosituimmat tietokantamoottorit ovat MyISAM ja tässä opinnäyteyössä käsiteltävä suomalaisen Innobase Oy:n kehittämä InnoDB. (Cabral, Murphy 2009, 375-376.)

Ennen versiota 5.5.5 MySQL:n oletustietokantamoottori oli MyISAM; nykyään tuon roolin on ottanut InnoDB (MySQL 2010c). MySQL:n järjestelmänvalvoja voi vaihtaa oletustietokantamoottoria muuttamalla default-storage-engine –muuttujaa my.cnf-konfiguraatiotiedostossa (MySQL 2010d).

Saman MySQL-tietokannan eri taulut voivat käyttää eri tietokantamoottoreita. Näin voidaan saavuttaa suorituskykyetua, mutta tähän sisältyy omat riskinsä. Useiden eri tietokantamootto-

reiden käyttäminen on rajattu tämän opinnäytetyön ulkopuolelle. Seuraavissa kappaleissa esitellään muutamia toisistaan olennaisesti poikkeavia tietokantamoottoreita.

2.2.1 InnoDB

Tässä opinnäytetyössä käsiteltävä InnoDB on suomalaisen InnoBase Oy:n kehittämä tietokantamoottori, joka on MySQL:n versiosta 5.5.5. lähtien myös MySQL:n oletustietokantamoottori (Wikipedia 2010a.) InnoBase on vuodesta 2005 lähtien ollut Oraclen tytäryhtiö (InnoBase 2010a).

InnoDB:n lisenssipolitiikka on samanlainen kuin MySQL:n (ks. MySQL) ja se on integroitu osaksi MySQL:n binäärejä. (Wikipedia 2010c.) InnoDB on suunniteltu erityisesti suurille datamäärille jolloin sen suorituskyky on optimaalisin. InnoDB tukee viiteavaimia ja transaktioita. (Cabral, Murphy 2009, 384.) Tarkempaa tietoa InnoDB:stä löytyy kappaleesta InnoDB:n toteutus.

2.2.2 MyISAM

MyISAM on riisuttu, kevyt tietokantamoottori joka ei tue transaktioita. Ennen MySQL:n versiota 5.5.5 se oli myös MySQL:n oletustietokantamoottori (MySQL 2010c). Se on suorituskyvyltään tehokas erityisesti tietoa luettaessa; tietoa kirjoitettaessa suorituskyky saattaa kärsiä sillä toisin kuin esim. InnoDB:ssä, lukitukset koskevat koko taulua eivätkä yksittäisiä rivejä. MyISAM-taulut varastoidaan kovalevylle kolmena eri tiedostona. Nämä tiedostot sisältävät koko taulun ja kaikki sen tiedot joten tiedostopohjainen varmuuskopiointi onnistuu helposti. (Cabral, Murphy 2009, 378-379.)

2.2.3 Falcon

Falcon on uuden sukupolven tietokantamoottori joka on suunniteltu erityisesti moniydinprosessoripalvelimille joissa on suuret määrät keskusmuistia. Suorituskyky on optimaalisin palvelimilla, joilla on vähintään 8 64-bittistä prosessoriydintä ja 8 gigatavua keskusmuistia. Falcon tukee transaktioita, rivitason lukituksia ja moniversiointia. (Cabral, Murphy 2009, 378, 401.)

2.2.4 Memory

Memory-tietokantamoottori varastoi nimensä mukaisesti kaiken tiedon palvelimen keskusmuistiin. Koska kovalevy on tietokoneen ylivoimaisesti hitain osa, ottamalla sen kokonaan

pois välistä Memory yltää salamannopeisiin luku- ja kirjoitusoperaatioihin. Kaikki tauluihin tehdyt muutokset häviävät uudelleenkäynnistyksen yhteydessä. Vaikka keskusmuistissa olevia tietoja on mahdollista tallentaa erilliseen tiedostoon, Memory ei ole käytännöllinen jos tauluihin tehtyjä muutoksia täytyy saada tallennettua uudelleenkäynnistyksestä toiseen. Tämän takia se soveltuukin parhaiten esim. sessiodatan käsittelyyn. Memory ei tue transaktiota, rivitason lukituksia eikä viiteavaimia. (Cabral, Murphy 2009, 378, 394-395.)

2.2.5 CSV

CSV-tietokantamoottori varastoi kaiken taulujen sisällön CSV (Comma-separated values) tekstitiedostoihin, joita käytetään yleensä siirtämään tietoa järjestelmästä toiseen. Taulujen sisältämää tietoa voi editoida tavallisella tekstieditorilla ja sen siirtäminen esim. Exceliin tai johonkin toiseen järjestelmään on erittäin helppoa. CSV varastoi kaiken tiedon kolmeen tiedostoon: .FRM-tiedostoon joka varastoi taulujen muodon, .CSM-tiedostoon johon varastoidaan metadata ja tavalliseen .CSV-tiedostoon johon varastoidaan taulujen sisältö. CSV-tietokantamoottori ei tue transaktioita eikä rivitason lukituksia. (Cabral, Murphy 2009, 378, 420.)

2.2.6 NDB

NDB on tietokantamoottori, jonka avulla on mahdollista luoda useasta MySQL Serveristä koostuva ”rypäs” nimeltään MySQL Cluster. Kaikki MySQL Clusterin tieto jaetaan usealle eri palvelimelle, jolloin pystytään estämään koko tietokannan kaatuminen tai tietojen katoaminen yhden palvelimen kaatuessa. Tieto replikoituu välittömästi kaikille MySQL Clusterin palvelimille; näin se eroaa tavallisesta asynkronisesta varmuuskopioiden ottamisesta. Jakamalla palvelu usealle palvelimelle voidaan saavuttaa tehokas suorituskyky myös edullisilla, suhteellisen tehottomilla palvelimilla. Järjestelmä on samasta syystä myös hyvin skaalautuva. NDB tukee transaktioita ja rivitason lukituksia. (Cabral, Murphy 2009, 378, 417.)

2.2.7 Yhteenveto

MySQL:n tietokantamoottori täytyy valita tietokannan (tai sen taulujen) käyttötarkoituksen mukaan. Kevyemmät tietokantamoottorit ovat yleensä nopeampia ainakin tietoa luettaessa, mutta ne eivät rajatumpien ominaisuuksiensa takia sovellu kaikkiin käyttötarkoituksiin. Seuraavassa taulukossa esitellään yhteenveto aiemmin esiteltyjen tietokantamoottoreiden keskeisimmistä ominaisuuksista.

Tietokantamoottori	Tuki transaktioille	Lukitustaso (rivi/taulu)	Tuki viiteavaimille
InnoDB	Kyllä	Rivi	Kyllä
MyISAM	Ei	Taulu	Kyllä*
Falcon	Kyllä	Rivi	Kyllä
Memory	Ei	Taulu	Ei
CSV	Ei	Taulu	Ei
NDB	Kyllä	Rivi	Kyllä

*Vain uusimmissa versioissa, aiemmin puuttunut.

Taulukko 1. Yhteenveto eri tietokantamoottoreiden keskeisistä ominaisuuksista (Cabral, Murphy 2009, 378.)

2.3 MySQL:n hallintatyökalut

Suurin osa MySQL-tietokannan hallinnasta tapahtuu mysqld (MySQL server daemon) -prosessille annettavien komentojen avulla. Näitä komentoja voidaan antaa komentoriviltä, joka on erittäin nopeaa ja tehokasta mutta vaatii käyttäjältä MySQL:n komentojen ja SQL-syntaksin hyvää tuntemusta. Toinen vaihtoehto MySQL:n hallitsemiseksi on graafisen käyttöliittymän sisältävä hallintatyökalu. Seuraavissa kappaleissa esitellään eri vaihtoehtoja MySQL:n hallinnoimiseen.

2.3.1 Komentorivi

Seuraavassa esitellään lyhyt esimerkki MySQL:n käyttämisestä komentorivin avulla. Esimerkissä kirjaudutaan sisään MySQL:ään, luodaan sinne uusi tietokanta, luodaan tähän tietokantaan yksi taulu ja lisätään kyseiseen tauluun muutama rivi tietoa. Esimerkissä on käytössä Debian GNU/Linux.

```
dbtech@debian:~$ mysql -u root -p          Kirjaudutaan sisään
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 39
Server version: 5.0.51a-24+lenny3 (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE test_db;        Luodaan uusi tietokanta test_db
Query OK, 1 row affected (0.02 sec)

mysql> SHOW DATABASES;                 Näytetään järjestelmän tietokannat
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
| test_db            |
+--------------------+
4 rows in set (0.01 sec)

mysql> USE test_db;                    Otetaan test_db-tietokanta käyttöön
Database changed
mysql> CREATE TABLE Test (             Luodaan tietokantaan uusi taulu
    -> id INT NOT NULL PRIMARY KEY,
    -> s VARCHAR(10),
    -> n INT DEFAULT 0
    -> ) TYPE=innodb;
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> START TRANSACTION;              Aloitetaan uusi transaktio ja lisätään tauluun muutama rivi
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Test (id,s,N) VALUES (1,'tekstiä',0);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Test (id,s,N) VALUES (2,'tekstiä',0);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;                         Kuitataan muutokset tietokantaan
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM Test;             Näytetään taulun sisältö
+----+----------+------+
| id | s        | n    |
+----+----------+------+
|  1 | tekstiä  |    0 |
|  2 | tekstiä  |    0 |
+----+----------+------+
2 rows in set (0.00 sec)
```

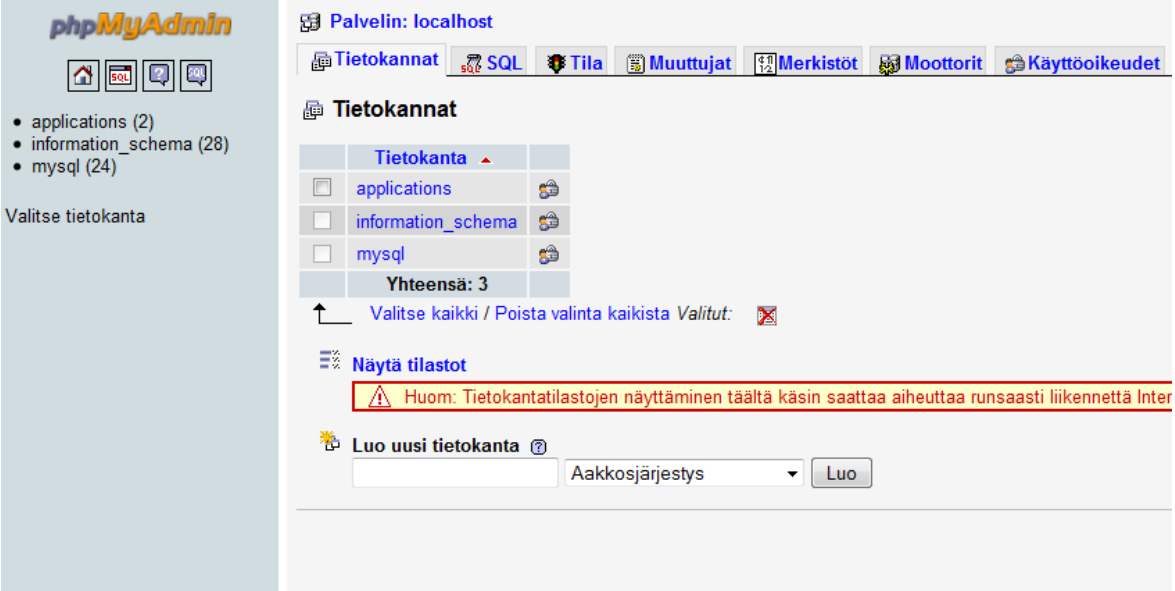
Kuvio 1. Näkymä phpMyAdmin-hallintatyökalusta.

Komentorivillä käytettävillä parametreilla on usein erikseen lyhyt ja pitkä muoto, esim. parametrit –u ja –p voidaan kirjoittaa myös muodossa --user ja --pass. Komentorivillä suoritettavien rutiinitehtävien määrää voi vähentää tallentamalla kyselyt tai asetukset erilliseen tiedostoon, joka voidaan suorittaa komentoriviltä. (Cabral, Murphy 2009, 50.)

2.3.2 phpMyAdmin

Yksi tunnetuimmista graafisista hallintatyökaluista on ilmainen, avoimen lähdekoodin phpMyAdmin, jonka avulla on mahdollista suorittaa suurin osa niistä operaatioista jotka onnistuvat komentoriviltäkin. phpMyAdmin sisältää myös käyttöliittymän minkä tahansa SQL-kyselyn

suorittamiseen. Kyseessä on PHP-kielellä kirjoitettu selainpohjainen työkalu, joten se vaatii toimiakseen WWW-palvelimen, esimerkiksi Apachen. PhpMyAdminin vahvuutena on se, että MySQL-tietokannan hallinta onnistuu mistä tahansa koneelta, jossa on internet-yhteys ja verkkoselain. (Cabral, Murphy 2009, 69.)



Kuvio 2. Näkymä phpMyAdmin-hallintatyökalusta.

2.3.3 SQLyog

SQLyog on hyvin tunnettu graafinen MySQL:n hallintatyökalu, joka mahdollistaa useimmat samat toiminnot kuin komentorivin käyttäminen. Näiden lisäksi se tarjoaa kehittyneitä tietokannan varmistus- ja synkronointitoiminnallisuuksia. SQL-kyselyitä voi joko kirjoittaa itse, jolloin IDE:n tyyppinen editori osaa itse ehdottaa oikeita avainsanoja tai ne voi rakentaa graafisesti, jolloin SQLyog rakentaa näitä vastaavat SQL-kyselyt. (Cabral, Murphy 2009, 66-68.)

SQLyog on saatavissa sekä ilmaisena avoimen lähdekoodin Community-versiona että maksullisina, suljetun lähdekoodin Professional, Enterprise ja Ultimate –versioina. Tuettuja käyttöjärjestelmiä ovat tällä hetkellä mm. Windows, Linux ja Mac OS X. (Wikipedia 2010d.)

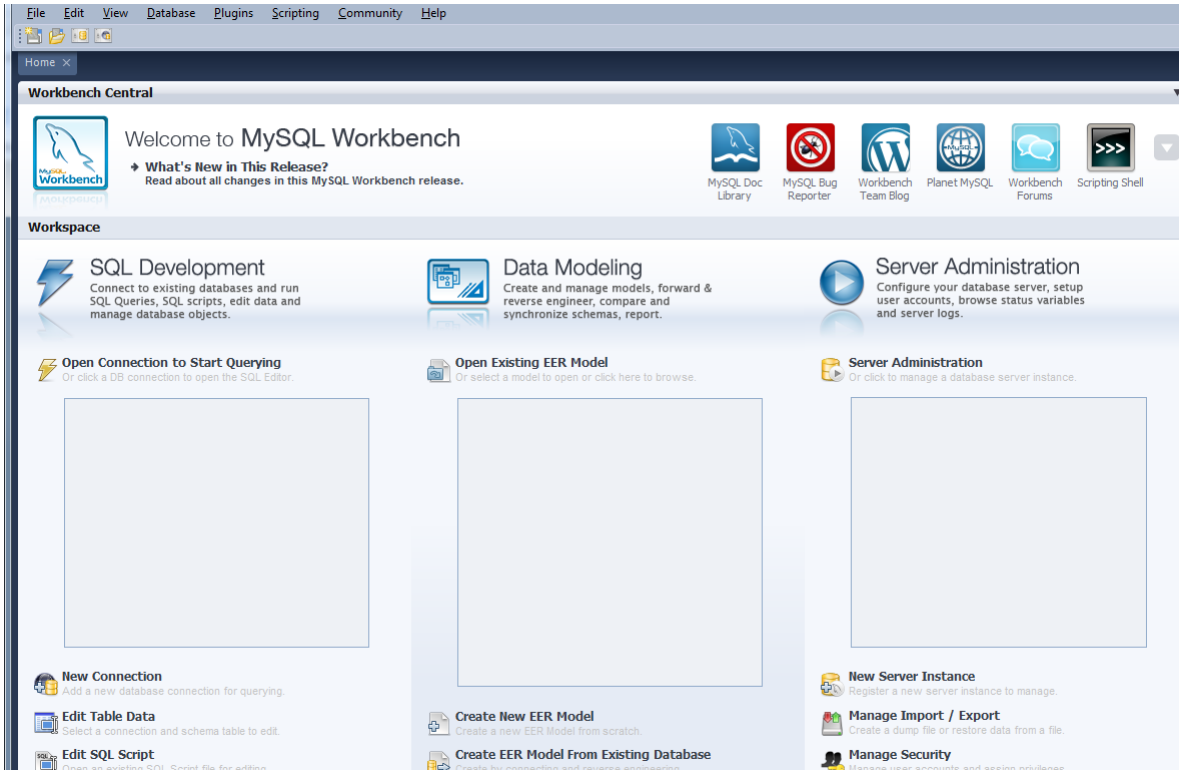
2.3.4 MySQL Workbench

MySQL Workbench on Oraclen kehittämä graafinen hallintatyökalu MySQL-tietokantoja varten. Se on saatavissa Community- ja Standard –versioina, joista ensimmäinen on maksuton. (MySQL 2010e.)

MySQL Workbench korvaa aiemman MySQL GUI Tools Bundlen, joka sisälsi mm. osat MySQL Workbench, MySQL Administrator ja MySQL Query Browser jotka on nyt integroitu osaksi uutta MySQL Workbenchia. (Wikipedia 2010e.)

Uudistetun MySQL Workbench-työkalun ominaisuudet on jaettu kolmeen osioon:

- SQL Development –osio mahdollistaa yhteyksien ja istuntojen määrittelyn, SQL-kyselyjen editoimisen sekä taulujen rakenteen ja sisällön muokkaamisen. (MySQL 2010f.)
- Data modeling -osio tarjoaa kehittyneitä työkaluja tietokannan mallintamiseen ml. tuen EER-kaavioille (enhanced entity-relationship diagram, kehittyneempi versio ER-kaaviosta). Näitä kaavioita voidaan takaisinmallintaa (eng. Reverse Engineering) olemassa olevista tietokannoista tai SQL-kyselyistä. (MySQL 2010g.)
- Server Administration –osioon on koottu tietokannan yleisten asetusten hallinnointi. Tämän osion kautta voi hallinnoida mm. käyttäjätilejä, instansseja, tietoturvaa, lokitiedostoja ja tietojen siirtämistä eri tietokantojen välillä. (MySQL 2010h.)



Kuvio 3. Näkymä MySQL Workbenchistä.

3 Teoreettinen tausta

3.1 Transaktiot

Transaktio on sarja tietokantaan suoritettavia operaatioita, jotka joko toteutetaan kokonaisuudessaan tai niitä ei toteuteta lainkaan. Klassinen esimerkki transaktiosta on tilisiirto. Jos tililtä A siirretään 100 € tilille B, täytyy sekä ensimmäisen operaation jossa tilin A saldosta vähennetään 100 € että toisen operaation jossa tilin B saldoon lisätään 100 € toteutua tai virhetilanteessa molempien jäädä toteutumatta. Tilanne, jossa tilitä A vähennettäisiin 100 €, mutta tietokannan tässä vaiheessa kaaduttua tilille B ei lisättäisi vastaavaa summaa johtaisi virheellisen tiedon syntymiseen. (Date, S. 2003, 76.)

MySQL:ssä transaktio alkaa START TRANSACTION –komennolla ja päättyy joko COMMIT-komentoon, joka kuittaa operaatioiden kokonaisuuden suoritetuksi tai ROLLBACK-komentoon, joka peruuttaa tehdyt muutokset kokonaan (näin transaktio ei jätä mitään muutoksia tietokantaan). (Cabral, Murphy 2009, 319-320.) Tämä periaate on määritelty SQL:2008-standardissa, mutta edellä mainittujen komentojen syntaksi vaihtelee eri tietokannanhallintajärjestelmien välillä.

Esimerkkitransaktio (pseudo-SQL):

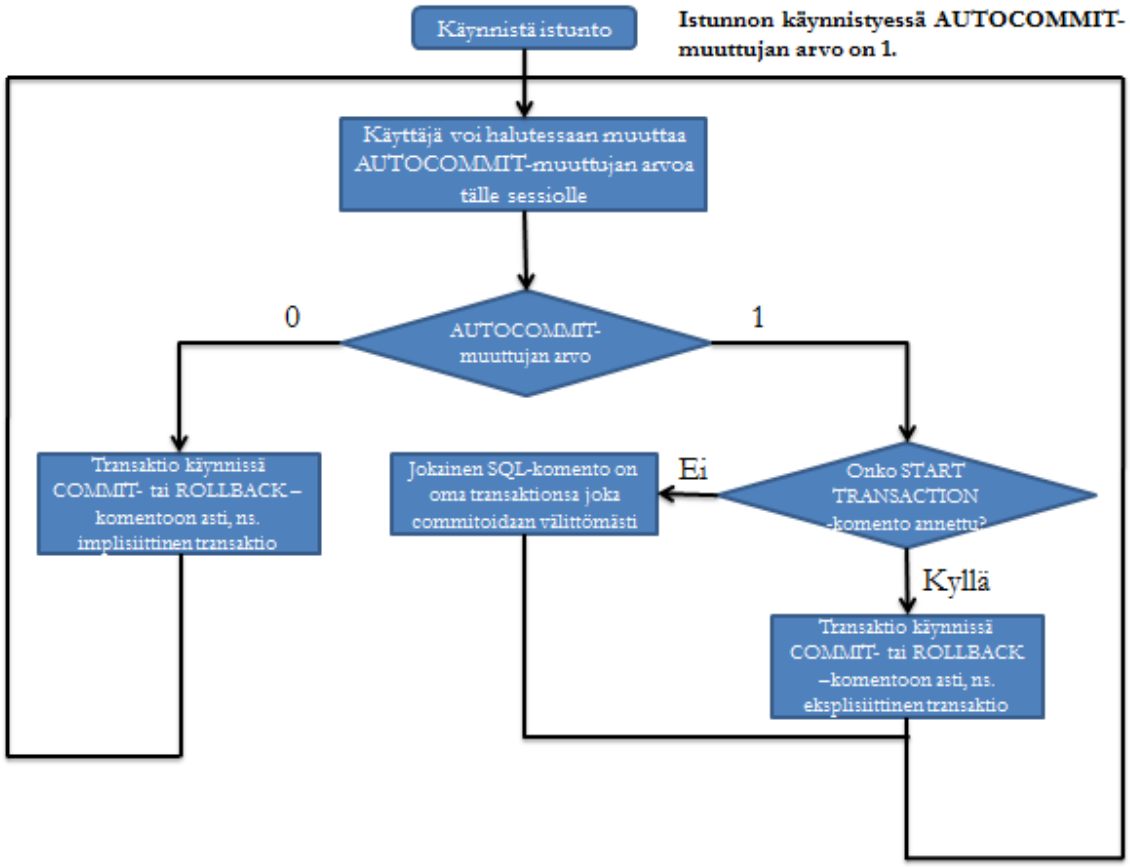
```
START TRANSACTION; -- komento, joka aloittaa uuden transaktion

UPDATE Tilit SET saldo = saldo - 100 -- vähennetään tilin A saldosta 100 euroa
WHERE tilinro = 1;

UPDATE Tilit SET saldo = saldo + 100 -- lisätään tilin B saldoon 100 euroa
WHERE tilinro = 2;

IF kaikki_onnistui -- tässä tarkistettaisiin, onko kaikki onnistunut
        THEN COMMIT; -- mikäli kaikki onnistui, kuitataan kaikki muutokset tietokantaan
        ELSE ROLLBACK; -- virhetilanteessa perutaan kaikki transaktion tekemät muutokset
END IF;
```

Kuvio 4. Esimerkkitransaktio

3.1.1 Implisiittiset ja eksplisiittiset transaktiot

Mikäli START TRANSACTION –komentoa ei ole, InnoDB käsittelee oletusarvoisesti jokaista SQL-komentoa omana transaktionaan, jolloin sen osoittamat muutokset kuitataan (COM-

MIT) välittömästi tietokantaan. Tätä käyttäytymistä voidaan muuttaa vaihtamalla AUTO-COMMIT-muuttujan arvoa. Esimerkiksi komennolla ”SET AUTOCOMMIT=0;” voidaan AUTOCOMMIT ottaa pois päältä jolloin komentoja ei kuitata toteutetuksi ennen COMMIT-komentoa (näin MySQL:ssä on aina transaktio päällä kunnes COMMIT- tai ROLLBACK-komento aloittaa seuraavan). Tällaiset transaktiot ovat ns. implisiittisiä. START TRANSACTION-komennon rooli on nimenomaan ottaa AUTOCOMMIT väliaikaisesti pois päältä. Tällä komennolla aloitetut transaktiot ovat ns. ekplisiittisiä. (Cabral, Murphy 2009, 319-320.)



Kuvio 5. Autocommit-moodin käyttäytyminen.

3.1.2 ACID

Akronyymi ACID kuvaa periaatteita, joita hyvin muodostetun transaktion täytyy noudattaa. Periaatteet ovat atomisuus (Atomicity), eheys (Consistency), eristyvyys (Isolation) ja pysyvyys (Durability). (Laiho, Laux 2010, 3-4.)

Atomisuus tai jakamattomuus tarkoittaa jo aiemmin mainittua periaatetta, jonka mukaan joko kaikki transaktioon kuuluvat operaatiot toteutetaan tai vaihtoehtoisesti jätetään kokonaan to-

teuttamatta. Edellä mainitussa esimerkissä tililtä A ei voi vähentää sataa euroa mikäli tilille B ei vastaavasti myös lisätä sataa euroa. (Laiho, Laux 2010, 3.)

Eheys tarkoittaa periaatetta, jonka mukaan transaktio vie tietokannan yhdestä ehyestä tilasta toiseen. Jos tietokanta oli ehyt ennen transaktion suorittamista, sen täytyy olla sitä myös transaktion suorittamisen jälkeen. Virhetilanteessa kaikki transaktion tekemät muutokset täytyy peruuttaa, jolloin transaktio ei jätä mitään muutoksia tietokantaan. SQL-standardin mukaan eheys ei ole välttämätöntä transaktion suorittamisen aikana, vain alku- ja lopputilanteessa. (Laiho, Laux 2010, 3.)

Eristyvyys tarkoittaa sitä, että transaktion tekemät muutokset täytyy piilottaa muilta samanaikaisesti suoritettavilta transaktioilta. Tämä eristyvyyden määrittely on saanut osakseen paljon kritiikkiä mm. tietokanta-asiantuntija Chris Datelta. Käytännössä nykyiset yleisimmät tietokannanhallintajärjestelmät voivat oikeita eristyvyystasoja käyttämällä estää transaktiota näkemästä muiden transaktioiden tekemiä muutoksia ja estää niitä kirjoittamasta tämän transaktion lukitsemien tietojen päälle, mutta ne eivät voi estää muita transaktioita lukemasta niiden luomaa ei-committoitua dataa. Tämä tilanne on mahdollinen esimerkiksi siinä tapauksessa, jos alin eristyvyystaso READ UNCOMMITTED on sallittu. (Laiho, Laux 2010, 3.)

Eristyvyystaso riippuu käytettävistä asetuksista; tiukempi eristyvyystaso saattaa hidastaa transaktioiden läpimenoaikaa ja heikentää suorituskykyä. Tätä aihetta käsitellään tarkemmin kappaleessa **Eristyvyystasot**.

Ihanteellisessa tilanteessa puhutaan transaktioiden **sarjallistumisesta (Serializability)**. Sarjallistumisessa transaktiot on suoritettu niin, että niiden lopputulos on vastaava kuin tilanteessa, jossa ne olisi suoritettu yksi kerrallaan. Tämä voi tilanteesta riippuen tapahtua myös vaikka käytössä ei olisikaan tiukin eristyvyystaso SERIALIZABLE.

Pysyvyys tarkoittaa periaatetta, jonka mukaan transaktion päätyttyä sen tulokset on tallennettu tietokantaan pysyvästi. Tämä koskee myös ohjelmiston tai laitteiston vikatilanteita. Tämän varmistaakseen tietokannanhallintajärjestelmät ylläpitävät lokitiedostoa, jota päivitetään jokaisen onnistuneen transaktion yhteydessä. Lokitiedostojen ja varmistusten avulla tietokanta voidaan tarvittaessa ”luoda uudelleen” ja palauttaa viimeisen onnistuneen transaktion jälkeiseen tilaan. (Laiho, Laux 2010, 4.)

3.2 Samanaikaisuuden hallinta (Concurrency control)

Samanaikaisuuden hallinnan tarkoituksena on varmistaa, että useat samanaikaisesti suoritettavat operaatiot tuottavat oikean tuloksen mahdollisimman hyvällä suorituskyvyllä (Wikipedia 2010f). Tässä opinnäytetyössä samanaikaisuuden hallinta viittaa teknologioihin, joilla pyritään varmistamaan, että tietokannanhallintajärjestelmässä samanaikaisesti suoritettavat prosessit (useimmiten transaktiot) eivät riko tietokannan eheyttä. Seuraavissa kappaleissa esitellään mahdollisia ongelmatilanteita, joita samanaikaisesti suoritettavat operaatiot voivat tietokannalle aiheuttaa.

3.3 Samanaikaisuuden hallinnan tyypilliset ongelmaskenaariot

SQL-standardissa määritellään neljä samanaikaisuuden hallinnan kannalta ongelmallista ilmiötä: Lost Update, Dirty Read, Non-Repeatable Read ja Phantom, jotka voidaan suomentaa esimerkiksi muotoon ”hukattu päivitys”, ”likainen luku”, ”muuttuva/pienenevä lukujoukko” ja ”kasvava lukujoukko” (Berenson, Bernstein, Gray, Melton, O'Neil & O'Neil 1995, 1). Nämä esitellään seuraavissa kappaleissa. Näiden lisäksi on tunnistettu muitakin samanaikaisuuden hallinnan ongelmaskenaarioita. Muut tässä opinnäytetyössä käsiteltävät ongelmaskenaariot esitellään kappaleessa **Muita samanaikaisuuden hallinnan ongelmaskenaarioita**.

3.3.1 Hukattu päivitys (Lost Update Problem)

Samanaikaisesti suoritettavat prosessit A ja B tekevät nostoja säästötililtä. Tilin alkusaldo haetaan muuttujaan jota käytetään uuden saldon päivittämiseen. Prosessi A tekee 100 euron noston säästötililtä. Prosessin A ollessa vielä kesken prosessi B nostaa samalta tililtä 200 euroa. Tililtä pitäisi näin vähentää 300 euroa, mutta prosessin B kirjoittaessa viimeisenä prosessin A tekemä päivitys hukataan ja tililtä vähennetäänkin vain 200 euroa. Hukatun päivityksen ongelma ei ole mahdollinen mikäli prosessit A ja B ovat transaktioita; jo alin SQL-standardissa määritetty eristyvyystaso READ UNCOMMITTED estää kyseisen ongelman syntymisen (ks. **Eristyvyystasot**). (Laiho, Laux 2010, 5, 9)

Esimerkki hukatusta päivityksestä ja kyseisen ongelmaskenaarion käyttäytyminen InnoDB:ssä on kuvattu liitteessä 1.

3.3.2 Likainen luku (Dirty Read Problem)

Transaktio A lukee tietoa, jota transaktio B on hieman aiemmin päivittänyt. Transaktio B päättyy kuitenkin ROLLBACK-komentoon joka palauttaa sen tekemät muutokset transaktiota edeltäneeseen tilaan. Näin transaktion A lukema tieto on virheellistä. Likainen luku –ongelma (kuten myös hukatun päivityksen ongelma) poistuu jos eristyvyystaso on vähintään vastaava kuin SQL-standardin READ COMMITTED (ks. **Eristyvyystasot**).. (Laiho, Laux 2010, 6, 9).

Esimerkki likaisesta lukemisesta ja kyseisen ongelmaskenaarion käyttäytyminen InnoDB:ssä on kuvattu liitteessä 2.

3.3.3 Muuttuva/pienenevä lukujoukko (Non-Repeatable Read Problem)

Muuttuvan tai pienevän lukujoukon ongelma syntyy tilanteessa, jossa transaktio A lukee tietoa tietokannan riveistä transaktion B samanaikaisesti muuttaessa tai poistaessa yhden tai useamman rivin tietoja. Kuvitellaan tilanne, jossa asiakkaalla X on kolme tiliä. Transaktio A lukee asiakkaan X tilien saldot. Transaktion A ollessa vielä käynnissä Transaktio B muuttaa jonkun asiakas X:n tileistä saldoa tai poistaa yhden tileistä kokonaan. Transaktion A lukiessa asiakkaan tilien saldot uudelleen se saa nyt eri tuloksen kuin ensimmäisellä lukukerralla. Lukuoperaatio ei ole toistettavissa (repeatable). (Laiho, Laux 2010, 6-7.)

Muuttuvan/pienenevän lukujoukon ongelma (kuten myös hukattu päivitys- ja likainen luku-ongelmat) poistuu jos eristyvyystaso on vähintään vastaava kuin SQL-standardin REPEATABLE READ. (ks. **Eristyvyystasot**). (Laiho, Laux 2010, 9.)

Muuttuva/pienenevä lukujoukko kuuluu samaan ristiriitaisen analyysin (Inconsistent Analysis) pääongelmaan kuin kasvava lukujoukko. (Laiho, Laux 2010, 6.)

Esimerkki muuttuvasta/pienenevästä lukujoukosta ja kyseisen ongelmaskenaarion käyttäytyminen InnoDB:ssä on kuvattu liitteessä 3.

3.3.4 Kasvava lukujoukko (Phantom Problem)

Kasvavan lukujoukon ongelma syntyy tilanteessa, jossa transaktio A lukee tietyn hakuehdon täyttävät rivit, jonka jälkeen samanaikaisesti suoritettava transaktio luo tauluun uuden rivin,

joka myös täyttäisi kyseisen hakuehdon. Jos transaktio ei tällä välin toista hakua, uusi rivi jää huomioimatta.

Kuvitellaan tilanne, jossa asiakkaalla X on kolme tiliä. Transaktio A lukee asiakas X:n tilien saldot. Transaktion A ollessa edelleen käynnissä transaktio B luo asiakkaalle X neljännen tilin, jossa on rahaa 5000 euroa. Tämä tili täyttäisi myös hakuehdon ”hae kaikki asiakkaan X tilit”. Mikäli transaktio A ei toista hakuoperaatiota, uusi tili jää huomioimatta, jolloin esim. transaktio A:n laskema asiakkaan X kaikkien tilien summa voisi olla 5000 euroa liian pieni. Uutta, huomioimatta jäänyttä tiliä kutsutaan tässä yhteydessä haamuriviksi (Phantom Row). Kasvavan lukujoukon ongelma (kuten myös hukattu päivitys, likainen luku- ja muuttuva/pienevä lukujoukko –ongelmat) poistuu jos eristyvyystasoksi on valittu tiukin eristyvyystaso SERIALIZABLE. (ks. **Eristyvyystasot**). (Laiho, Laux 2010, 9.)

Kasvava lukujoukko kuuluu samaan ristiriitaisen analyysin (Inconsitent Analysis) pääongelmaan kuin muuttuva/pienenevä lukujoukko. (Laiho, Laux 2010, 6.)

Esimerkki kasvavasta lukujoukosta ja kyseisen ongelmaskenaarion käyttäytyminen InnoDB:ssä on kuvattu liitteessä 4.

3.4 Eristyvyystasot (Isolation levels)

SQL-standardi on versiosta SQL-92 lähtien määritellyt seuraavat eristyvyystasot ratkaisuksi edellä mainittuihin samanaikaisuuden hallinnan ongelmaskenaarioihin. Teoriassa ja usein myös käytännössä tiukempi eristyvyystaso huonontaa suorituskykyä joten sopiva eristyvyystaso täytyy valita tapauskohtaisesti.

Ongelma: Eristyvyystaso:	Hukattu päivitys (Lost Update)	Likainen luku (Dirty Read)	Muuttuva/pienenevä lukujoukko (Non-Repeatable Read)	Kasvava lukujoukko (Phantom)
READ UNCOMMITTED	Ei mahdollista	Mahdollista	Mahdollista	Mahdollista
READ COMMITTED	Ei mahdollista	Ei mahdollista	Mahdollista	Mahdollista
REPEATABLE READ	Ei mahdollista	Ei mahdollista	Ei mahdollista	Mahdollista
SERIALIZABLE	Ei mahdollista	Ei mahdollista	Ei mahdollista	Ei mahdollista

Taulukko 2. Eristyvyystasot (Laiho, Laux 2010, 9.)

Käytännössä nämä eristyvyystasot ovat saaneet osakseen paljon kritiikkiä (Berenson ym. 1995, 1). Todelliset tietokannan hallintajärjestelmien toteuttamat eristyvyystasot saattavat poiketa näistä määrittelyistä melkoisesti, mutta esimerkiksi SQL Server ja DB2 toteuttavat kaikki neljä SQL-standardin eristyvyystasoa tietyin poikkeuksin.

Tietokannassa voidaan yleensä käyttää samanaikaisesti useita eri eristyvyystasoja käyttäviä transaktioita. Näin voidaan parantaa suorituskykyä, sillä esim. lukevat transaktiot eivät yleensä tarvitse yhtä tiukkaa eristyvyystasoa kuin tietoa päivittävät transaktiot. SQL-standardin mukaan eristyvyystaso tulee asettaa jokaiselle transaktiolle erikseen (MySQL:ssä SET TRANSACTION ISOLATION LEVEL –komento) mutta sitä ei tulisi muuttaa transaktion aloittamisen jälkeen. Käytännössä eristytyystaso on joissain tietokannanhallintajärjestelmissä mahdollista muuttaa myös transaktion aloittamisen jälkeen.

3.5 Muita samanaikaisuuden hallinnan ongelmaskenaarioita

3.5.1 Ristiinpäivittävä UPDATE-UPDATE –kilpailu

Tämä skenaario pohjautuu Martti Laihon ja Fritz Laux'n tutorialiin On SQL Concurrency Technologies. Tässä skenaariossa transaktiot A ja B päivittävät kahta (tässä tapauksessa saman taulun) riviä eri järjestyksessä. Transaktio A päivittää riviä 1 ensimmäisenä, Transaktio B taas päivittää ensimmäisenä riviä 2. Tämän jälkeen molemmat transaktiot yritetään commitoida. Mikäli transaktio A ja B odottavat molemmat vastakkaiselta transaktiolta vapautuvaa lukkoa, kyseessä on lukkiumatilanne (deadlock) jonka tietokannanhallintajärjestelmän (MySQL:n tapauksessa tietokantamoottorin) täytyy ratkaista jollain tavalla. (Laiho, Laux 2010, 45)

Ristiinpäivittävä UPDATE-UPDATE –kilpailu ja sen käyttäytyminen InnoDB:ssä on kuvattu liitteessä 5.

3.5.2 Isärivin poistaminen (Delete parent)

Tietokannan tauluille on yleensä määritelty viite-eheyssääntöjä, jotka varmistavat esimerkiksi sen, että tietokannan lapsirivi ei voi jäädä ilman isäriviä. (Hovi 2010, 107.)

Kuvitellaan esimerkiksi Tili-taulu, jossa omistajan id –sarake osoittaa jokaisen tilin kuuluvan tietylle omistajalle (isä-lapsi –suhde). Ongelmia seuraa esimerkiksi sellaisessa tilanteessa, jossa transaktio A hakee SELECT-lauseella tietyn omistajan tilien yhteissaldon ja transaktion ollessa

vielä käynnissä transaktio B poistaa kyseisen omistajan tietokannasta. Koska omistajalla ja tilillä on isä-lapsi –suhde, pitäisi viiteavaimen eheysmäärityksistä riippuen joko omistajien poistamisen olla estetty (RESTRICT) tai kaikkien kyseisen omistajan tilien poistua Tilit-taulusta (CASCADE): Jälkimmäisessä vaihtoehdossa SELECT-lauseen hakema tieto muuttuisi vääräksi eikä olisi toistettavissa. Lisäksi on olemassa muita viite-eheysvaihtoehtoja jotka tuskin tulisivat kysymykseen tämän tyyppisessä sovelluksessa.

Esimerkki isärivin poistamisesta ja kyseisen ongelmaskenaarion käyttäytyminen InnoDB:ssä on kuvattu liitteessä 6.

3.5.3 Rakenteen muuttaminen

Peter Gulutzan ja Trudy Pelzer kuvaavat teoksessaan: **SQL-99 Complete, Really (1999, 700)** periaatteen, jonka mukaan tietokannan rakennetta muuttavia lauseita ei tyypillisesti tulisi suorittaa samanaikaisesti muiden SQL-lauseiden kanssa.

Väärin ajoitettu rakenteen muuttaminen seuraa esimerkiksi tilanteessa, jossa transaktio A on hakenut tietoa SELECT-lauseella ja sen ollessa vielä käynnissä transaktio B yrittää muuttaa kyseisen taulun rakennetta pudottamalla sarakkeen, johon transaktio A viittaa. Transaktion A:n SELECT-lauseen hakema tieto on muuttunut vääräksi (saraketta ei ole enää olemassa) eikä kyseinen lause ole toistettavissa.

Esimerkki rakenteen muuttamisesta ja kyseisen ongelmaskenaarion käyttäytyminen InnoDB:ssä on kuvattu liitteessä 7.

3.6 Samanaikaisuuden hallinnan teknologiat (Concurrency technologies)

Edellä mainitut SQL-92 –standardissa määritellyt eristyvyystasot eivät ota kantaa tietokannanhallintajärjestelmien todellisuudessa toteuttamiin samanaikaisuuden hallinnan teknologioihin. Suurin osa tämän päivän tietokannanhallintajärjestelmistä käyttää samanaikaisuuden hallinnan toteuttamiseen joko lukituksia tai moniversiointia (Wikipedia 2010f.)

3.6.1 Lukitus (Locking scheme concurrency control)

Lukitukset ovat yleisin samanaikaisuuden hallinnan ongelmien ratkaisemiseen käytetty tekniikka. Ennen relaatiotietokantoja samanaikaisuuden hallinta perustui kokonaan lukituksiin. (Laiho, Laux 2010, 15.)

Yksi varhaisimmista lukitustekniikoista on **luku- ja kirjoituslukkojen** käyttäminen. Tässä tekniikassa on määritelty kaksi erilaista lukkoa, **S-lukko (jaettu lukko, eng. Shared lock)** ja **X-lukko (yksinomainen lukko, eng. Exclusive lock)**. Näiden lisäksi useissa lukitustekniikoissa on määritetty **U-lukko (päivitykseen varautumislukko, eng. Update queuing lock)** Oikein muodostettu transaktio voi lukea tietoa vain niistä tietoalkioista, joihin se on saanut S-lukon, ja se voi kirjoittaa vain niihin tietoalkioihin, joihin sillä on X-lukko. (Laiho, Laux 2010, 15.)

S-lukon (Shared lock) tarkoituksena on varmistaa, ettei mikään muu transaktio muuta kyseisen tietoalkion tietoa ennen kuin lukkoa pyytänyt transaktio on suoritettu. Usealla transaktiolla voi olla S-lukko samaan tietoalkioon, tästä on peräisin sen nimitys jaettu lukko. (Laiho, Laux 2010, 15.)

X-lukon (Exclusive lock) tarkoituksena on lukita tietoalkio kirjoittamista varten niin, ettei mikään muu transaktio voi lukea sitä (tieto muuttuisi vääräksi heti tiedon päivittymisen hetkellä) tai kirjoittaa sen päälle mitään. Vain yhdellä transaktiolla kerrallaan voi olla X-lukko tietoalkioon, siksi sitä kutsutaankin yksinomaiseksi lukoksi. (Laiho, Laux 2010, 15.)

U-lukko (Update queuing lock) myönnetään esim. SQL Serverissä ja DB2:ssa ensimmäiselle X-lukkoa pyytävälle transaktiolle jolle X-lukkoa ei voi juuri sillä hetkellä myöntää. Se voidaan myöhemmin korottaa X-lukoksi. U-lukko on ikään kuin jonossa odottamassa korotusta X-lukoksi, mistä juontaa sen nimi päivitykseen varautumislukko. (Laiho, Laux 2010, 15.) U-lukot suunniteltiin estämään **lukkiuma (deadlock) -**nimisiä tilanteita, jossa kaksi transaktiota estää toistensa etenemisen. Lukkiuma seuraa esimerkiksi silloin kun kaksi transaktiota on hankkinut S-lukon samaan tietoalkioon. Jos molemmat transaktiot yrittävät seuraavaksi päivittää samaa tietoalkiota, ne yrittävät korottaa S-lukkonsa X-lukoksi. Tämä ei kuitenkaan onnistu kummaltakaan, sillä vastakkaisen transaktion asettama S-lukko estää X-lukon hankkimisen tietoalkiolle. Näin molemmat transaktiot joutuisivat teoriassa odottamaan ikuisesti kun kumpikaan transaktioista ei voi edetä. Nykyisissä tietokannanhallintajärjestelmissä tämä ongelma on ratkaistu valitsemalla toinen transaktioista ”uhriksi”, joka peruutetaan ROLLBACK-komennolla. Peruute-

tusta transaktiosta annetaan virheilmoitus johon tietokantaa käyttävä sovellus voi reagoida kaappaamalla sen poikkeukset käsittelevällä rakenteella (esim. Javassa try-catch). (Laiho, Laux 2010, 19.)

Pyydetty lukko:	Toiselle prosessille samaan resurssiin myönnetty lukko			
	Ei ole	S	U	X
S	Salli	Salli	Salli	Odota
U	Salli	Salli	Odota	Odota
X	Salli	Odota	Odota	Odota

Taulukko 3. S-, U- ja X-lukkojen yhteensopivuus muiden lukkojen kanssa. (Laiho, Laux 2010, 15.)

Lukitukset ovat usein joko rivi- tai taulukohtaisia. InnoDB, kuten muutkin MySQL:n raskaammat transaktiopohjaiset tietokantamoottorit käyttävät sekä rivi- että taulukohtaista lukitusta. MyISAM ja monet muut tietokantamoottorit taas käyttävät taulupohjaista lukitusta. (Cabral, Murphy 2009, 378.) Lukitus voi joissain tietokannanhallintajärjestelmissä kohdistua myös johonkin muuhun yksikköön, kuten sivuun (tietty määrä fyysisesti peräkkäistä dataa tallennusmedialla, esim. 8 kt) tai extenttiin (tietty määrä peräkkäisiä sivuja, esim. 8) (Laiho, Laux 2010, 16)

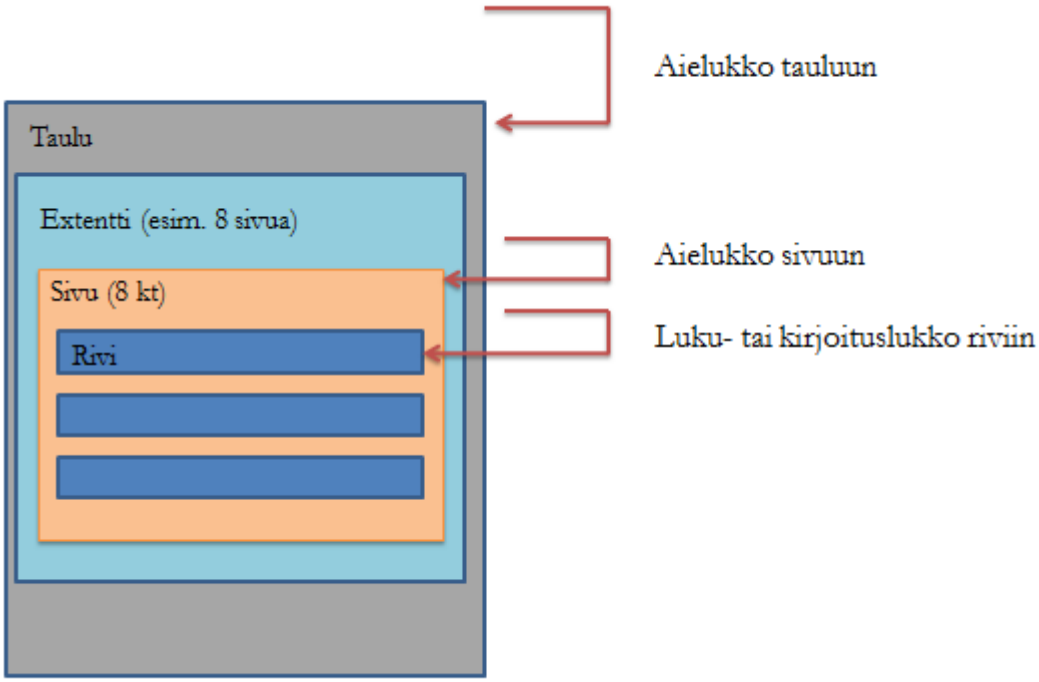
Nykyisten tietokannanhallintajärjestelmien käyttämät lukitukset ovat usein huomattavasti edellä mainittua monimutkaisempia.

3.6.2 Monirakeinen lukitus (Multi-granular locking scheme concurrency control)

Monirakeinen lukitus (Multi-granular locking scheme) perustuu samoihin periaatteisiin kuin luku- ja kirjoituslukkojen käyttäminen, mutta lukkoja on useampia ja ne voivat kohdistua usealle eri karkeustasolle (näitä voivat olla esim. rivi, sivu, extentti tai taulu). Monirakeinen lukitus on kehittyneempi lukitusteknologia joka myös muistuttaa enemmän yleisimpien tietokannanhallintajärjestelmien todellisia toteutuksia. Usean tietokannanhallintajärjestelmän (mm. SQL Serverin ja DB2:n) käyttämät lukitukset ovat nimenomaan monirakeisia (Laiho, Laux 2010, 15-16.)

Monirakeisessa lukituksessa käytetään useita eri lukkoja, joista lukemiseen tarkoitetut jaetut S-lukot ja kirjoittamiseen tarkoitetut yksinomaiset X-lukot esiteltiin jo aiemmassa kappaleessa (ks. Luku- ja kirjoituslukot). Näiden lisäksi on yleensä toteutettu ainakin aielukot IS, IX ja SIX.

Aielukot eivät vielä mahdollista lukemista tai kirjoittamista vaan niiden avulla tarkistetaan mahdollisuuksia saada S- tai X-lukko alemmalla tasolla. Kaikkien aielukkojen tarkoitus on saada lopulta S- tai X-lukko pienimmälle mahdolliselle tasolle, joka voi asetuksista riippuen olla esim. rivi tai sivu. (Laiho, Laux 2010, 16.) Mitä pienemmälle tasolle lukitus kohdistuu, sitä vähemmän konfliktitilanteita esiintyy. Toisaalta tällöin tarvitaan enemmän palvelimen resursseja, kuten suoritintehoa, keskusmuistia ja kovalevytilaa useampien lukkojen käsittelemiseen. (Microsoft 2010a).



Kuvio 6. Esimerkki monirakeisen lukituksen hierarkiasta. (Laiho, Laux 2010, 17.)

Transaktion yrittäessä saada S-lukon tietoalkioon, se hankkii ensin IS-lukon (Intent shared lock) ylemmän tason rakenteeseen. Jos tämä onnistuu, pyydetään aina uusi IS-lukko seuraavalle alemman tason rakenteelle (jokin tasoista saatetaan järjestelmästä riippuen jättää väliin) kunnes päästään halutulle tasolle, esim. riviin. Halutun tason rakenteelle voidaan nyt ottaa S-lukko. X-lukon tapauksessa menetellään samalla tavalla hankkimalla ensin IX-lukko (Intent exclusive lock) ylemmän tason rakenteelle. SIX-lukko (Shared with intent exclusive lock) rakenteelle hankitaan tilanteessa, jossa transaktio aikoo lukea kaikkia kyseisen rakenteen alle kuuluvia rakenteita ja kirjoittaa joihinkin näistä (joihin tullaan hankkimaan erikseen IX-lukot). (Laiho, Laux 2010, 16-17.) Seuraavassa taulukossa kuvataan lukkojen yhteensopivuus toistensa kanssa.

Pyydetty lukko:	Toiselle prosessille samaan resurssiin myönnetty lukko:				
	IS	IX	S	SIX	X
IS	Salli	Salli	Salli	Salli	Odota
IX	Salli	Salli	Odota	Odota	Odota
S	Salli	Odota	Salli	Odota	Odota
SIX	Salli	Odota	Odota	Odota	Odota
X	Odota	Odota	Odota	Odota	Odota

Taulukko 4. Monirakeisen lukituksen lukkojen yhteensopivuus muiden lukkojen kanssa (Laiho, Laux 2010, 17.)

3.6.3 Aikaleimaus (Timestamp ordering)

Aikaleimaus on samanaikaisuuden hallinnan teknologia, joka perustuu lukitusten sijaan transaktioille ja tietoalkioille annettaviin aikaleimoihin (timestamp). Transaktion aikaleima osoittaa sen alkamisajankohtaa. Jokaisen näistä on oltava yksilöllinen, kahta samalla aikaleimalla varustettua transaktiota ei saa esiintyä. Tietoalkioilla on kaksi aikaleimaa, lukuaikaleima (read timestamp) joka osoittaa, milloin tietoalkion tietoa on viimeksi luettu ja kirjoitusaikaleima (write timestamp) joka osoittaa, milloin sen sisältämää tietoa on viimeksi päivitetty. (Wikipedia 2010g.)

Transaktion yrittäessä lukea tietoa tietoalkiosta tarkistetaan, osoittaako tietoalkion kirjoitusaikaleima myöhäisempää ajankohtaa kuin transaktion aikaleima. Tässä tapauksessa transaktio on peruutettava ja aloitettava alusta, koska jokin toinen transaktio on muuttanut tietoalkion sisältämää dataa transaktion käynnistämisen jälkeen. Jos näin ei ole, lukuoperaatio voidaan suorittaa loppuun ja tietoalkion lukuaikaleimaksi annetaan transaktion aikaleima. (Wikipedia 2010g.)

Transaktion yrittäessä kirjoittaa tietoalkioon, tarkistetaan sekä tietoalkion luku- että kirjoitusaikaleima. Jos lukuaikaleima osoittaa myöhäisempää ajankohtaa kuin transaktion aikaleima, täytyy transaktio peruuttaa ja käynnistää uudelleen, sillä jokin muu operaatio on oletettavasti ottanut kopion tietoalkion sisältämästä datasta transaktion käynnistämisen jälkeen ja päälle kirjoittaminen tekisi tästä datasta väärää. Jos taas tietoalkion kirjoituslukuleima osoittaa myöhäisempää ajankohtaa, transaktiota ei käynnistetä uudelleen mutta kirjoitusoperaatio jätetään väliin. Muussa tapauksessa kirjoitusoperaatio voidaan suorittaa loppuun ja tietoalkion kirjoitusaikaleimaksi annetaan transaktion aikaleima. (Wikipedia 2010g.)

Mikään nykypäivän yleisimmistä tietokannanhallintajärjestelmistä ei käytä aikaleimausta joten sitä ei käsitellä tässä opinnäytetyössä enempää.

3.6.4 Moniversiointi (Multi-version timestamp ordering)

Moniversiointi (MVCC) on lukituksien lisäksi toinen yleisessä käytössä oleva samanaikaisuuden hallinnan teknologia (Wikipedia 2010f). Myös moniversiointi perustuu aikaleimoihin (kuten tavallinen aikaleimaus), mutta sen toimintaperiaate on erilainen. MVCC perustuu siihen, että tietokannanhallintajärjestelmä (tai tietokantamoottori) ylläpitää useita versioita samoista tietoalkioista. (Laiho, Laux 2010, 21.) Kuten tavallisessa aikaleimauksessa, transaktiolla on oma yksilöllinen aloitusajankohtaa kuvaava aikaleimansa, ja jokaisella tietoalkiolla on sekä kirjoitus- että lukuaikaleima (ks. **Aikaleimaus**).

Transaktion A yrittäessä lukea tietoalkiosta X tietoa se saa käyttöönsä sen X:n version, jolla on uusin mahdollinen kirjoitusaikaleima joka ei kuitenkaan osoita myöhäisempää aikaa kuin transaktion oma aikaleima, eli uusimman mahdollisen version jonka päivitys ei tapahtunut transaktion A alkamisen jälkeen. (Laiho, Laux 2010, 22; Cabral, Murphy 2009, 335.)

Transaktion A jälkeen alkava transaktio B tekee samoin. Jos transaktio A nyt päättyy COMMIT-komentoon (jolloin X päivitetään), B ei enää ”näe” X:ää. Se voi työskennellä vain sen datan kanssa mitä sillä on käytettävissään. Ainoa tapa, millä transaktio B saa tietoalkion X uudelleen näkyviin, on transaktion peruuttaminen (ROLLBACK) ja käynnistäminen uudelleen. (Cabral, Murphy 2009, 335.)

Transaktio voi päivittää tietoalkiota (eli luoda siitä uuden version), mikäli transaktion aikaleima osoittaa myöhäisempää ajankohtaa kuin tietoalkion luku- ja kirjoitusaikaleimat. Jos näin ei ole, kirjoitusoperaatio ei onnistu ja transaktio täytyy peruuttaa (ROLLBACK) ja aloittaa uudelleen. Jos kirjoitusoperaatio onnistuu, tietoalkiosta luodaan uusi versio jolle annetaan luku- ja kirjoitusaikaleimoiksi transaktion aikaleima. (Laiho, Laux 2010, 22.)

3.6.5 Optimistinen samanaikaisuudenhallinta (Optimistic concurrency control)

Optimistinen samanaikaisuudenhallinta (OCC) on samanaikaisuuden hallinnan teknologia, joka perustuu sille olettamukselle, että samanaikaisesti suoritettavat transaktiot päätyvät vain harvoin konfliktitilanteisiin. Konflikteiden tarkastus tehdään vasta lopun COMMIT-komennon yhteydessä, jonka jälkeen konfliktitilanteeseen joutuneet transaktiot peruutetaan

(ROLLBACK). Transaktion suorittamisen aikana kaikki sen suoraan tietokantaan tekevät operaatiot ovat lukuoperaatioita. Kirjoitusoperaatiot tehdään paikallisille kopioille ja niiden synkronointi tietokantaan tapahtuu vasta konfliktien tarkastuksen jälkeen. (Laiho, Laux 2010, 22.)

Jos konfliktitilanteita on myös todellisuudessa harvoin, saattaa optimistinen samanaikaisuudenhallinta johtaa suorituskyvyn parantumiseen. Toisaalta konfliktitilanteissa suorituskyky kärsii huomattavasti enemmän kuin käytettäessä muita samanaikaisuuden hallinnan teknologioita. (Wikipedia 2010h.) Sen tietokantakirjallisuudessa saamasta suuresta julkisuudesta huolimatta optimistista samanaikaisuudenhallintaa käyttävät tietokannanhallintajärjestelmät ovat yhä melko harvinaisia. Yksi tällaisista järjestelmistä on Länsi-Skotlannin yliopiston kehittämä Pyrrho DBMS. (Laiho, Laux 2010, 22.)

4 Kilpailevat tietokannanhallintajärjestelmät

Yleisimmät yritysmaailmassa käytössä olevat tietokannanhallintajärjestelmät ovat Oracle, DB2 ja SQL Server. Ne kaikki perustuvat E.F Coddin vuonna 1970 kehittämään relaatiomalliin ja tukevat SQL-kyselykieltä; tuotepohjainen SQL saattaa tosin poiketa standardista. Uusien tietokannanhallintajärjestelmien kehittäminen ja kyseisten järjestelmien vaihtaminen vaatii paljon resursseja, joten näiden ns. ”kolmen suuren” valta-aseman uskotaan kestävän pitkälle tulevaisuuteen. (Hovi 2010, 2, 5, 265-266.)

4.1 Oracle

Oracle Database (myöhemmin: Oracle) on Oracle Corporationin kehittämä tietokannanhallintajärjestelmä. Sen ensimmäinen versio julkaistiin vuonna 1979 ja se oli aikanaan myös ensimmäinen kaupallinen SQL:ää käyttävä tietokanta. (Wikipedia 2010i.)

Oracle tukee SQL:n lisäksi myös PL/SQL:ää (Procedural Language/Structured Query Language), joka on Oraclen kehittämä proseduraalinen laajennus SQL:ään. Se muistuttaa tavallista proseduraalista ohjelmointikieltä tarjoten tuen muuttujille, ehtolauseille, silmukoille ja poikkeusten käsittelylle. PL/SQL perustuu standardoituun ADA-ohjelmointikieleen. (Zeis, Ruel & Wessler 2009, 125-126.)

Oracle käyttää samanaikaisuuden hallintaan lukitusten ja moniversioinnin sekoitusta. Ainoat sallitut eristyvyystasot ovat READ COMMITTED ja SERIALIZABLE, jotka poikkeavat hieman SQL-standardin samannimisistä eristyvyvyystasoista. Oletuseristyvyystaso on READ COMMITTED, joka muistuttaa toiminnaltaan DB2:n eristyvyystasoa CURRENTLY COMMITTED. Oracle tukee sekä rivi- että taulutason lukituksia ja mahdollistaa lisäksi eksplisiittisen rivien tai taulujen lukitsemisen käyttäen LOCK IN SHARE MODE- tai FOR UPDATE –vaihtoehtoa SELECT-lauseissa. (Laiho, Laux 2010, 32-34.)

Oraclen uusin versio 11g on jaettu ominaisuuksien perusteella Enterprise, Standard, Standard One ja Lite –editioihin. Edellisestä 10g versiosta on saatavilla myös ilmainen Express-versio. Oracle on saatavissa usealle eri käyttöjärjestelmälle, mm. Windowsille, Linuxille ja Mac OS X:lle. (Wikipedia 2010i.)

4.2 DB2

DB2 on IBM:n kehittämä tietokannanhallintajärjestelmä. Sen juuret ulottuvat jo 70-luvun alkuun ja siihen aikaan IBM:llä työskennelleen relaatiomallin luojan, Edgar F. Coddin ajatuksiin. IBM (joka on myös SQL:n alkuperäinen kehittäjä) julkaisi DB2:n ensimmäisen kaupallisen version vuonna 1983. (Wikipedia 2010j.)

DB2 tukee SQL:n lisäksi myös SQL PL:ää (SQL Procedural Language), joka on IBM:n kehittämä proseduraalinen laajennus SQL:ään. SQL PL tarjoaa tuen muuttujille, ehtolauseille, silmukoille ja poikkeusten käsittelylle. (IBM 2010a.)

DB2:n samanaikaisuuden hallinta perustuu monirakeiseen lukitukseen. Nämä lukitukset kohdistuvat oletusarvoisesti riveihin, mutta ne voidaan konfiguroida taulukohtaisiksi. Osa editioista tukee myös sivukohtaisia lukituksia. DB2:ssa on määritelty eristyvyystasot REPEATABLE READ, READ STABILITY, CURSOR STABILITY ja UNCOMMITTED READ jotka eroavat jossain määrin SQL-standardin eristyvyystasoista. Versiosta 9.7 lähtien DB2:een lisättiin uusi eristyvyystaso CURRENTLY COMMITTED, joka on myös uusi oletuseristyvyystaso. CURRENTLY COMMITTED vastaa toiminnaltaan Oraclen READ COMMITTED -eristyvyystasoa. (Laiho, Laux 2010, 26; IBM 2010b.)

DB2:n uusin versio 9.7 on jaettu ominaisuuksien perusteella Express-C, Workgroup Server ja Enterprise Server –editioihin. Express-C -editio on ilmainen. DB2 on saatavissa usealle eri käyttöjärjestelmälle, mm. Windowsille, Linuxille ja IBM i:lle. (Wikipedia 2010j.)

4.3 SQL Server

Microsoft SQL Server (myöhemmin SQL Server) on Microsoftin kehittämä tietokannanhallintajärjestelmä. SQL Serverin pohjautuu alun perin Microsoftin entisen yhteistyökumppanin Sybasen samannimiseen tuotteeseen, jota nykyään markkinoidaan nimellä Adaptive Server Enterprise. (Wikipedia 2010k.)

SQL Serverin kanssa kommunikoimiseen käytetään pääasiassa T-SQL (Transact-SQL) –kieltä, joka on Microsoftin kehittämä proseduraalinen laajennus SQL:ään. T-SQL tarjoaa tuen muuttujille, ehtolauseille, silmukoille ja poikkeusten käsittelylle. (Schneider, Gibson 2008, 315-316.)

SQL Server käyttää samanaikaisuuden hallintaan monirakeista lukitusta, joka muistuttaa kappaleessa **Monirakeinen lukitus** esitettyä teoriaa. SQL Serverin toteutuksessa on kuitenkin otettu käyttöön useampia lukitustasoja ja lukkoja. SQL Server tarjoaa kaikki neljä SQL-standardin määrittelemää eristyvyystasoa. Oletuseristyvyystaso on READ COMMITTED. Yksittäinen SQL Server -tietokanta voidaan konfiguroida käyttämään moniversiointia, jolloin saadaan käyttöön uusi eristyvyystaso Snapshot (Microsoft 2010b; Laiho, Laux 2010, 23-24,52.) Lisäksi SQL Server tarjoaa transaktioille Deadlock tuning priority –ominaisuuden (lukuarvo -10:n ja +10:n välillä), jonka avulla voidaan asettaa eri transaktioille prioriteetteja sen mukaan, minkä niistä halutaan selviytyvän voittajana lukkiumatilanteessa (Laiho, Laux 2010, 25).

SQL Serverin uusin versio (SQL Server 2008 R2) on jaettu ominaisuuksien perusteella Parallel Data Warehouse, Datacenter, Enterprise, Standard, Workgroup, Web, Embedded, Fast Track, Compact, Express ja Developer –editioihin. Express- ja Developer –editiot ovat ilmaisia. SQL Server on saatavissa ainoastaan Windows-käyttöjärjestelmälle. (Wikipedia 2010k.)

5 InnoDB:n toteutus

Merkittävä osa InnoDB:n arkkitehtuurista ja algoritmeista perustuu Jim Grayn ja Andreas Reuterin teokseen "Transaction Processing: Concepts and Techniques". Samanaikaisuuden hallinta InnoDB:ssä perustuu sekä lukituksiin että moniversiointiin (Oracle 2010; Innobase 2010). Nämä käsitellään lyhyesti tässä kappaleessa.

InnoDB tarjoaa kaikki neljä SQL-standardin määrittelemää eristyvyystasoa (ks. Eristyvyystasot). Oletuseristyvyystaso on REPEATABLE READ, joka on SQL-standardia vertailukohtana käyttäen yhtä tasoa tiukempi kuin Oraclen, DB2:n ja SQL Serverin oletuseristyvyystasot. (MySQL 2010i.)

InnoDB:ssä on käytössä neljä erilaista lukkoa: S, X, IS ja IX, jotka toimivat samalla tavalla kuin kappaleessa **Monirakeinen lukitus** mainitut vastaavat. Lukot S ja X kohdistuvat rivitasolle, aielukot IS ja IX taas kohdistuvat tauluihin. Saadakseen S-lukon riville täytyy transaktion saada ensin IS-lukko vastaavalle taululle. Samoin saadakseen X-lukon riville täytyy transaktion ensin saada IX-lukko vastaavalle taululle. (MySQL 2010j.)

Pyydetty lukko:	Toiselle prosessille samaan resurssiin myönnetty lukko:			
	X	IX	S	IS
X	Odota	Odota	Odota	Odota
IX	Odota	Salli	Odota	Salli
S	Odota	Odota	Salli	Salli
IS	Odota	Salli	Salli	Salli

Taulukko 5. InnoDB:n lukkojen yhteensopivuus muiden lukkojen kanssa. (MySQL 2010j.)

Edellä mainittujen lukkojen lisäksi InnoDB:ssä voidaan lukita myös mm. indeksivälejä (Gap lock). Lisäksi indeksi- ja rivitason lukituksia yhdistetään ns. seuraavan avaimen lukoksi (Next-key lock) (MySQL 2010i.).

InnoDB käyttää niin sanottuja ryvästettyjä indeksejä (Clustered Indexes), jolloin tietokannan data on järjestetty fyysisesti tietyn uniikin indeksin, yleensä pääavaimen mukaan. Tämä nopeuttaa hakuja, joissa haetaan tällaisen indeksin tietyn arvoalueen sisälle kuuluvaa tai muuten sen kanssa samassa järjestyksessä olevaa dataa. (MySQL 2010k; Wikipedia 2010l.)

InnoDB on moniversioiva tietokantamoottori, joten sen täytyy säilyttää useita versioita tietokannan taulujen riveistä (ks. Moniversiointi). Tämä johtaa suurempaan kovalevytilan tarpeeseen (MySQL 2010l). Eristyvyystasoilla READ COMMITTED ja REPEATABLE READ InnoDB käyttää oletuksena ns. eheitä ei-lukitsevia lukuoperaatioita (Consistent Nonlocking Reads), joiden avulla lukuoperaatioita voidaan suorittaa ilman lukituksia ottamalla tietokannasta ns. tilannevedos (Snapshot), joka vastaa tietyn ajankohdan tilannetta tietokannassa. (MySQL 2010m.) Niihin tilanteisiin, joihin tämä ei sovellu, voidaan käyttää eksplisiittistä lukitusta SELECT-lauseen yhteydessä. SELECT ... FOR UPDATE estää muita transaktioita päivittämästä tai lukemasta kyseistä tietoalkiota kuten X-lukko, kun taas SELECT ... LOCK IN SHARE MODE antaa S-lukon tavoin muiden transaktioiden lukea kyseisten tietoalkioiden dataa mutta estää näihin tehtävät päivitykset. Kokonaisia tauluja voidaan lukita LOCK TABLES –komennolla ja näin luotu lukitus voidaan myöhemmin poistaa UNLOCK TABLES –komennolla. (MySQL 2010n.)

Autocommit-moodi on InnoDB:ssä automaattisesti päällä, jolloin jokainen SQL komento käsitellään oletusarvoisesti omana transaktionaan (ks. **Transaktiot**). Useampia SQL-komentoja voidaan kuitenkin myös Autocommit-moodin ollessa päällä ryhmitellä yhdeksi transaktioksi käyttämällä eksplisiittistä START TRANSACTION –komentoa. (MySQL 2010i; MySQL 2010o.) Autocommitin ollessa pois päältä on InnoDB:ssä aina transaktio käynnissä, kunnes COMMIT- tai ROLLBACK-komento aloittaa seuraavan. Tällaiset transaktiot ovat ns. implisiittisiä transaktioita.

6 Tutkimustavat

Pääasiallinen tässä tutkimuksessa käytetty tutkimustapa on InnoDB:n käyttäytymisen seuraaminen ennalta määritellyissä samanaikaisuuden hallinnan skenaarioissa. Tutkimuksen tuloksissa saatetaan viitata InnoDB:n dokumentaatioon, mutta kaikki tulokset pyritään verifioimaan empiiristen kokeiden avulla. Kukin skenaario on jaettu vaiheisiin siten, että jokainen SQL-lause muodostaa oman vaiheensa. Kustakin vaiheesta kuvataan MySQL:n antamat ilmoitukset, InnoDB:n ottamat lukot (lukkojen monitorointia varten asennetaan InnoDB Plugin –niminen lisäosa) sekä tarvittaessa tietokannan väliaikainen tila. Jokaisesta skenaariosta kuvataan aina sen tietokantaan aikaansaama lopputila.

Skenaarioita tutkitaan kahdella MySQL-istunnolla, joiden lisäksi käytetään kolmatta istuntoa tietokannan tilan tutkimiseen. Molemmilla samanaikaisilla transaktioilla on sama eristyvyystaso. Kustakin skenaariosta kuvataan vähintäänkin InnoDB:n antamat ilmoitukset, sen ottamat lukot ja tulokset enintään neljällä eli eristyvyystasolla. Kaikkia eristyvyystasoja ei välttämättä kokeilla mikäli InnoDB:n käyttäytyminen näillä eristyvyystasoilla on itsestään selvää. Kustakin skenaariosta saattaa muodostua yksi tai useampia alaskenaarioita mikäli InnoDB:n käyttäytyminen herättää uusia kysymyksiä. Lisäksi skenaarioita kokeillaan tarvittaessa myös muilla tietokannanhallintajärjestelmillä mikäli halutaan saadaa lisää tietoa näiden eroavaisuuksista.

Tutkimuksessa käytetyt skenaariot on pääsääntöisesti koostettu Martti Laihon ja Fritz Laux'n artikkelista "On SQL Concurrency Technologies" mutta niitä on tarpeen mukaan muokattu siten, että ne soveltuvat paremmin InnoDB:llä tutkittavaksi.

7 Tulokset

Tutkimuksen pääasiallisena tarkoituksena oli tutkia, miten InnoDB käyttäytyy seuraavissa etukäteen määritellyissä samanaikaisuuden hallinnan ongelmaskenaarioissa:

- Hukatun päivityksen ongelma (Lost Update Problem)
- Likainen luku (Dirty Read Problem)
- Muuttuva lukujoukko (Non-Repeatable Read)
- Kasvava lukujoukko (Phantom Problem)
- Ristiinpäivittävä UPDATE-UPDATE -kilpailu
- Isärivin poistaminen (Delete Parent Problem)
- Rakenteen muuttaminen

Tuloksia tutkittaessa on huomioitava, että InnoDB näyttää asettamansa lukot vain siinä tapauksessa, että niihin kohdistuu jokin konflikti. Esimerkiksi riville asetettu S-lukko näytetään vasta silloin, kun jokin toinen transaktio yrittää saada samaan riviin kohdistuvaa X-lukkoa. Kyseinen S-lukko on olemassa jo tätä ennen, mutta sen tiedot näytetään vasta toisen transaktion pyytäessä X-lukkoa.

7.1 Eristyvyystasojen (Isolation Levels) toteutus InnoDB:ssä

InnoDB:n tarjoamat eristyvyystasot READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ ja SERIALIZABLE muistuttavat oleellisilta osin SQL-standardin vastaavia, joskin tämä riippuu osin käytettävistä asetuksista.

Tutkimuksessa todettiin, että InnoDB käyttää oletusarvoisesti moniversiointia eristyvyystasoilla READ COMMITTED ja REPEATABLE READ ja lukituksia eristyvyystasolla SERIALIZABLE. Eristyvyystasolla READ UNCOMMITTED transaktio hakee aina ensisijaisesti ensimmäisen ei-commitoidun tiedon ja toissijaisesti uusimman commitoidun tiedon tietokannasta. Tämä rikkoo perinteistä ACID-periaatetta sillä toisilla transaktioilla ei ole mitään keinoja estää tällaista transaktiota lukemasta keskeneräistä tietoa tietokannasta.

7.2 Moniversioinnin toiminnasta InnoDB:ssä

Moniversiointi on InnoDB:ssä oletusarvoisesti käytössä eristyvyystasoilla READ COMMITTED ja REPEATABLE READ. Käytännössä tämä tarkoittaa sitä, että käytössä ovat ns. eheät ei-lukitsevat lukuoperaatiot (Consistent Nonlocking Reads), jotka nimensä mukaisesti eivät käytä lukituksia. Eristyvyystasolla READ COMMITTED transaktio hakee tietokannasta SELECT-lausetta suoritettaessa uusimman commitoidun version. Mikäli SELECT-lause toistetaan, haetaan jälleen sen hetkinen uusin commitoitu versio, joka saattaa olla eri kuin ensimmäisellä kerralla. Eristyvyystasolla REPEATABLE READ transaktio saa joka kerta SELECT-lauseelleen saman tuloksen, jonka se sai suorittaessaan tämän lauseen ensimmäistä kertaa (poikkeuksena ainoastaan taulun tai sarakkeen pudottaminen, jonka transaktio ”näkee” välittömästi). Mikäli kyseisestä tiedosta halutaan uudempi versio, täytyy transaktio käynnistää uudelleen antamalla sille ensin COMMIT- tai ROLLBACK –komento.

Moniversiointi voidaan ottaa pois päältä ja näiden sijaan käyttää lukituksia myös eristyvyystasoilla READ COMMITTED ja REPEATABLE READ. Tämä tapahtuu käyttämällä SELECT-lauseen yhteydessä LOCK IN SHARE MODE- tai FOR UPDATE- vaihtoehtoa. LOCK IN SHARE MODE –vaihtoehtoa käytettäessä kyseiselle tietoalkiolle yritetään saada S-lukko, FOR UPDATE –vaihtoehtoa käytettäessä yritetään puolestaan saada X-lukko. Nämä lukot myönnetään, mikäli ne eivät aiheuta konfliktia jonkin aiemman lukon kanssa. (ks. liite 9).

7.3 Skenaarioiden tulokset

7.3.1 Hukatun päivityksen ongelma (Lost Update Problem)

Hukattuun päivitykseen liittyvät testit löytyvät liitteestä 1.

Pääskenaariossa kuvattua hukattua päivitystä (lost update) ei esiintynyt edes alimmalla READ UNCOMMITTED –eristyvyystasolla (ks. pääskenaario). Tämä on SQL-standardin mukaista. Antamalla molempien transaktioiden COMMIT-komennot saatiin kuitenkin aikaiseksi tilanne, jossa toisen transaktion tekemä päivitys katoaa (ks. alaskenaario 1). Ongelma esiintyi tasoilla READ UNCOMMITED, READ COMMITTED ja REPEATABLE READ. Vasta tiukin eristyvyystaso SERIALIZABLE poisti ongelman.

Pääskenaariota kokeiltiin vielä SERIALIZABLE –eristyvyystasolla jotta saataisiin selville, kuinka pitkälle kyseinen skenaario etenee. Selvisi, että edes toisen transaktion muuttujiin ha-

kemaa tietoa ei saa muuttumaan vanhaksi sillä InnoDB estää sellaisen tiedon muuttamisen, josta jokin toinen käynnissä oleva transaktio on hakenut tietoa SELECT-lauseella.

Edellä mainittu skenaario päättyi lock timeoutiin transaktion A jäädessä odottamaan lukon saamista. Kokeiltiin vielä luoda lukkiuma (deadlock) antamalla myös transaktion B UPDATE-komento, jolloin molemmat transaktiot odottaisivat toiselta transaktiolta vapautuvaa lukkoa (ks. alaskenaario 2). InnoDB selvisi tilanteesta valitsemalla transaktion B uhriksi jolloin transaktio A pääsi suoriutumaan loppuun asti. Tietokannan eheys ei kärsinyt tilanteessa.

7.3.2 Likainen luku (Dirty Read Problem)

Likaiseen lukuun liittyvät testit löytyvät liitteestä 2 .

Pääskenaariossa kuvattu likainen luku onnistui SQL-standardin mukaisesti READ UNCOMMITTED mutta ei READ COMMITTED tai sitä tiukemmalla eristyvyystasolla (ks. pääskenaario). Jos skenaarion kulkua kuitenkin muutettiin niin, että transaktion B haettua tietoa muuttujaan antaakin transaktio A ROLLBACK-komennon sijaan COMMIT-komennon, muuttujan sisältämä tieto muuttui vanhaksi eristyvyystasoilla READ COMMITTED ja REPEATABLE READ (ks. alaskenaario 1). Testi paljastaa InnoDB:n käyttävän moniversiointia näillä eristyvyystasoilla. Eristyvyystaso SERIALIZABLE ratkaisee ongelman estämällä sellaisen tiedon hakemisen muuttujaan, johon on tehty ei-commitoituja muutoksia. Tiedon hakeminen estetään tässä tapauksessa rivikohtaisella X-lukolla.

7.3.3 Muuttuva/pienenevä lukujoukko (Non-Repeatable Read)

Muuttuvaan/pienenevään lukujoukkoon liittyvät testit löytyvät liitteestä 3.

Pääskenaariossa kuvattu muuttuva lukujoukko –ongelma esiintyi eristyvyystasolla READ COMMITTED mutta ei eristyvyystasoilla REPEATABLE READ- tai SERIALIZABLE. Tämä on SQL-standardin mukaista. Eristyvyystasolla REPEATABLE READ transaktio B onnistui kuitenkin muuttamaan transaktio A:n SELECT-lauseella hakeman tiedon vanhaksi. Eristyvyystasolla SERIALIZABLE InnoDB estää myös tämän ongelman esiintymisen.

7.3.4 Kasvava lukujoukko (Phantom Problem)

Kasvavaan lukujoukkoon liittyvät testit löytyvät liitteestä 4.

Pääskenaariossa kuvattu kasvava lukujoukko onnistui eristyvyystasolla REPEATABLE READ mutta ei eristyvyystasolla SERIALIZABLE (ks. pääskenaario). Tämä on SQL-standardin mukaista. Jälkimmäinen eristyvyystaso käyttää haamurivien estämiseen ns. indeksivälilukkoa (GAP Lock).

Alaskenaariossa 1 tutkittiin indeksivälilukon käyttäytymistä käyttämällä laajempaa testiaineistoa. Transaktio A laski tietyn omistajan tilien lukumäärän. Testissä selvisi, että InnoDB oli asettanut indeksivälilukon (S, GAP) tämän omistajan id:lle 4 sekä S-lukot tätä id:tä edeltävälle id:lle 3. Transaktio B onnistui lisäämään tilejä Omistaja-taulun ensimmäiselle omistajalle 2 ja taulun viimeiselle omistajalle 5. Tässä oli kuitenkin se poikkeus, että omistajan 5 uuden tilin id ei voinut olla pienempi kuin tämän omistajan sen hetkisten tilien id:t. (ks. alaskenaario 1). Tämä liittyy mahdollisesti InnoDB:n käyttämiin ns. ryvästettyihin indekseihin (Clustered Indexes, ks. luku InnoDB:n toteutus).

7.3.5 Ristiinpäivittävä UPDATE-UPDATE –kilpailu

Ristiinpäivittävään UPDATE-UPDATE –kilpailuun liittyvät testit löytyvät liitteestä 5.

Pääskenaariossa transaktiot A ja B päivittävät kahta (tässä tapauksessa saman taulun) riviä eri järjestyksessä. Transaktio A päivittää riviä 1 ensimmäisenä, Transaktio B taas päivittää ensimmäisenä riviä 2. Transaktiot jäävät odottamaan lukkoa siihen riviin, jota ne päivittävät toisena. Transaktioiden annetaan odottaa lukkoa timeoutiin asti. Tämän jälkeen molemmat transaktiot yritetään commitoida. Skenaario johtaa lopputulokseen, jossa kumpikin transaktioista onnistuu commitoimaan tietokantaan sen rivin muutoksen, jonka sisältöä ne päivittävät ensimmäisenä. Tämä viittaa siihen, että lock timeoutin tapauksessa koko transaktiota ei peruuteta (ROLLBACK), vaan ainoastaan kyseinen epäonnistunut UPDATE-komento peruuntuu (ks. pääskenaario). Tällainen lopputulos saattaa rikkoa tietokannan eheyden. Timeoutin tapauksessa sovelluksen tulisi reagoida tilanteeseen antamalla ROLLBACK-komento. InnoDB kyllä ehdottaa tätä (”try restarting transaction”), mutta suostuu silti commitoimaan tiedot mikäli COMMIT-komento annetaan.

Alaskenaariossa 1 annetaan myös transaktion B UPDATE-komento kun transaktio A odottaa vielä lukon saamista. Tämä johtaa lukkiumaan (deadlock), jonka InnoDB ratkaisee valitsemalla transaktion B uhriksi jolloin transaktio A pääsee jatkamaan loppuun asti. Transaktio B commitoidaan tämän jälkeen ensimmäisenä, mutta se ei jätä mitään muutoksia tietokantaan. Tämä

viittaa siihen, että lukkiumatilanteessa uhriksi joutuneelle transaktiolle annetaan ROLLBACK-komento. Transaktio A puolestaan voittaa molempien rivien, myös jälkimmäisenä päivittämänsä rivin päivityksen ja commitoi muutokset tietokantaan. (ks. alaskenaario 1).

7.3.6 Isärivin poistaminen (Delete Parent Problem)

Isärivin poistamiseen liittyvät testit löytyvät liitteestä 6.

Pääskenaariossa kuvattu isärivin poistaminen onnistui kaikilla eristyvyystasoilla lukuun ottamatta tiukinta eristyvyystasoa SERIALIZABLE. Kaikilla muilla eristyvyystasoilla esiintyi jokin samanaikaisuuden hallinnan ongelma. READ UNCOMMITED- ja READ COMMITED –eristyvyystasoilla transaktio A sai toistamalleen SELECT-lauseelle tuloksen NULL. Ero oli (kuten voidaan olettaa) ainoastaan siinä, että READ UNCOMMITTED näki tehdyn muutoksen heti ja READ COMMITTED vasta siinä vaiheessa, kun kyseinen muutos oli commitoitu tietokantaan. REPEATABLE READ –eristyvyystasolla transaktio A:n SELECT-lause palautti joka kerta saman tuloksen kuin ensimmäisellä kerralla; myös sen jälkeen kun kyseiset rivit olivat kokonaan poistettu tietokannasta.

Eristyvyystaso SERIALIZABLE esti sellaisten rivien poistamisen, joihin jokin toinen transaktio on kohdistanut SELECT-lauseen. Tämä tapahtui asettamalla S-lukko erikseen jokaiselle riville, joihin SELECT-lause kohdistui.

7.3.7 Rakenteen muuttaminen

Rakenteen muuttamiseen liittyvät testit löytyvät liitteestä 7.

Pääskenaariossa kuvattu rakenteen muuttaminen onnistui kaikilla eristyvyystasoilla lukuun ottamatta tiukinta eristyvyystasoa SERIALIZABLE. Kaikilla muilla eristyvyystasoilla transaktio A:n toistama SELECT-lause kaatui heti, kun transaktio B oli poistanut enimi-sarakkeen (jo siinä vaiheessa kun tätä muutosta ei oltu vielä commitoitu tietokantaan). Poikkeuksellisinta tässä se, että tämä ilmiö tapahtui myös eristyvyystasolla REPEATABLE READ, jolla SELECT-lauseen tulisi aina olla toistettavissa (repeatable) samalla tuloksella. Sarakkeen tai taulun pudottaminen tekee tässä ilmeisesti poikkeuksen. Eristyvyystasolla SERIALIZABLE sarakkeen poistaminen ei onnistu johtuen transaktion A omistajaan id = 2 hankkimasta S-lukosta. (ks. pääskenaario)

7.4 Muita huomioita

Tutkimuksessa todettiin, että InnoDB sallii transaktion eristyvyystason muuttamisen kesken transaktion suorittamisen (ks. Liite 8). Eristyvyystaso voidaan muuttaa miksi tahansa (niin tiukemmaksi kuin löyhemmäksikin) eristyvyystasojen ominaisuuksien pysyessä ennallaan.

InnoDB:n todettiin odottavan lukon saamista oletusarvoisesti 50 sekuntia ennen lock timeoutia. Lock timeoutin pituutta voidaan muuttaa muuttamalla innodb_lock_wait_timeout-muuttujan arvoa konfiguraatiotiedostossa. (MySQL 2010p.)

Lukkiumatilanteista (deadlock) ei seurannut ongelmia. InnoDB valitsi aina johdonmukaisesti toisen transaktiosta uhriksi jolloin vastakkainen transaktio pääsi jatkamaan loppuun asti. InnoDB:n dokumentaation mukaan uhriksi pyritään valitsemaan transaktioista ”pienempi”, toisin sanoen se, joka on lisännyt, poistanut tai päivittänyt harvempaa riviä. (MySQL 2010q.)

8 Johtopäätökset

Tutkimuksen pääasiallisena tarkoituksena oli tutkia, miten InnoDB käyttäytyy ennalta määritellyissä samanaikaisuuden hallinnan skenaarioissa (ks. **Tavoitteet ja tutkimusongelmat**).

Tutkimuksessa todettiin, että InnoDB:n samanaikaisuuden hallinta ei suoraan muistuta Oraclen, DB2:n tai SQL Serverin samanaikaisuuden hallintaa. Oletusasetuksilla InnoDB käyttää Oraclen tapaan moniversiointia joka voidaan tarpeen vaatiessa ottaa pois SELECT…LOCK IN SHARE MODE tai SELECT…FOR UPDATE –lauseella. InnoDB tarjoaa kuitenkin DB2:n ja SQL Serverin tapaan kaikki neljä SQL-standardin tarjoamaa eristyvyystasoa. Eristyvyystasolla SERIALIZABLE InnoDB käyttää monirakeista lukitusta joka muistuttaa, mutta on kuitenkin erilainen kuin DB2:n ja SQL Serverin vastaavat. InnoDB:n monirakeinen lukitus perustuu vain kahteen lukitustasoon (rivi ja taulu), joiden lisäksi voidaan lukita mm. indeksivälejä. Huomion arvoista on myös se, että tutkimuksessa käytetyillä työkaluilla InnoDB:n asettamat lukot saatiin näkyviin vasta siinä vaiheessa, kun ne olivat konfliktissa jonkin toisen pyydetyn lukon kanssa.

InnoDB on tuotteena huomattavasti uudempi kuin mikään edellä mainituista, joten InnoDB:n kehittäjät ovat ilmeisesti pyrkineet poimimaan InnoDB:tä varten parhaita olemassa olevia samanaikaisuuden hallinnan teknologioita. Samanaikaisuuden hallinnan kannalta InnoDB onkin erittäin monipuolinen tuote mm. edellä mainituista syistä johtuen.

Lisäksi tutkimuksessa oli tarkoitus tutkia, miten InnoDB on toteuttanut SQL-standardin mukaiset eristyvyystasot. Testeissä eristyvyystasojen todettiin noudattavan oleellisilta osin standardin määrityksiä, tosin eristyvyystasoilla READ COMMITTED ja REPEATABLE READ transaktion käyttäytyminen riippuu siitä, käytetäänkö moniversiointia. Testeissä 1-4 tutkittiin miten eristyvyystasot käyttäytyvät ilmiöiden: hukattu päivitys (Lost Update), likainen luku (Dirty Read), muuttuva/pienenevä lukujoukko (Non-Repeatable Read) ja kasvava lukujoukko (Phantom) tapauksessa. Näiden testien pääskenaarioissa eristyvyystasot käyttäytyivät SQL-standardin mukaisesti, mutta testejä muokkaamalla saatiin aikaiseksi mm. toisen transaktion tekemän päivityksen katoamista tai muuttujaan haetun tiedon muuttumista vanhaksi kaikilla eristyvyystasoilla lukuun ottamatta tiukinta eristyvyystasoa SERIALIZABLE.

Testeistä pystyttiin päättelemään, että transaktion jäädessä odottamaan lukon vapautumista lock timeoutiin asti ei koko transaktiota peruuteta (ROLLBACK), vaan ainoastaan kyseinen lukkoa odottanut komento. Mikäli sovellus ei reagoi tähän, saattaa seurauksena olla rikkonainen tilanne tietokannassa kun kahdesta transaktiosta kumpikin onnistui päivittämään kahdesta yrittämästään rivistä vain toisen. MySQL ehdottaa transaktion käynnistämistä mutta jättää päätöksen sovellukselle. Myös COMMIT-komento sallitaan vaikka osa transaktiosta on epäonnistunut.

InnoDB sallii tietokannan rakennetta muuttavien DDL-lauseiden suorittamisen samanaikaisesti tietokannan dataa muuttavien DML-lauseiden kanssa. Oletusasetuksilla tämä johtaa samanaikaisuuden hallinnan ongelmatilanteisiin kaikilla muilla eristyvyystasoilla lukuun ottamatta tiukinta eristyvyystasoa SERIALIZABLE. Muilla eristyvyystasoilla InnoDB sallii sellaisten tietoalkioiden poistamisen, johon jonkin käynnissä olevan transaktion SELECT-lause viittaa. Tämä johtuu ilmeisesti siitä, että kyseiset SELECT-lauseet eivät käytä lukituksia näillä eristyvyystasoilla. Toistettu SELECT-lause saattoi palauttaa joko tuloksen ”NULL” tai tuloksen, jota ei enää ole olemassa tietokannassa. Kokonaisen taulun poistamisen tapauksessa saatiin jopa aikaiseksi tilanne, jossa eristyvyystasoa REPEATABLE READ käyttävä transaktio palautti nimestään huolimatta SELECT-lauseen toistaessaan eri tuloksen kuin ensimmäisellä kerralla. Eristyvyystasoilla READ COMMITTED ja REPEATABLE READ tilanne voidaan korjata lukitsemalla kyseiset rivit tai taulut SELECT…LOCK IN SHARE MODE- tai SELECT…FOR UPDATE –lauseella.

Tiukin eristyvyystaso SERIALIZABLE todettiin tutkimuksessa täysin vedenpitäväksi. Laajoista testeistä huolimatta tälle eristyvyystasolla ei saatu aikaiseksi ainuttakaan samanaikaisuuden hallinnan ongelmaa.

Tutkimuksessa saatiin aikaiseksi laajat ja kattavat testit InnoDB:n käyttäytymisestä samanaikaisuuden hallinnan ongelmatilanteissa. Kaikkien testien läpikäyminen muilla tietokannanhallintajärjestelmillä osoittautui mahdottomaksi testien lukumäärän kasvaessa liian suureksi, joten testeissä päädyttiin lopulta testaamaan InnoDB:n toimintaa syvällisemmin jättäen muut tietokannanhallintajärjestelmät vähemmälle huomiolle. Myöskään suorituskykyä ei otettu tutkimuksessa huomioon. Voidaan kuitenkin todeta, että oikein konfiguroituna InnoDB selviää samanaikaisuuden hallinnan ongelmaskenaarioista niin, ettei tietokannan eheys kärsi tai haettu tieto muutu vääräksi.

Lisäksi tutkimuksen aikana selvisi varotoimenpiteitä, joita sovelluskehittäjien tulisi noudattaa kehittäessään sovelluksia MySQL/InnoDB –yhdistelmän päälle. Esimerkiksi ohjelmoijien tietämys tietokannoista saattaa olla rajallista joten annetut neuvot tuskin ovat itsestään selviä.

8.1 Jatkotutkimusehdotukset

Mahdollisia jatkotutkimuksen aiheita syntyy sitä enemmän, mitä laajempaa kokonaisuutta halutaan tutkia. Pelkästään InnoDB:stä voitaisiin tutkia samanaikaisuuden hallintaa, kun käytössä on useita eri eristyvyystasoja käyttäviä transaktioita tai samanaikaisuuden hallintaa useammalla kuin kahdella samanaikaisella transaktiolla. Laajentamalla tutkimusaluetta muihin MySQL:n tietokantamoottoreihin voitaisiin tutkia esimerkiksi samanaikaisuuden hallintaa tietokannassa, jossa on käytössä useampi kuin yksi tietokantamoottori. Nämä aiheet eivät ole kaikissa tietokannoissa merkityksellisiä mutta niillä saatetaan saada suorituskykyetua tietokannoissa, joissa sekä tietokannan koko että samanaikaisten transaktioiden määrä ovat suuria.

Mikäli näitä aiheita päätetään jatkotutkia, on tutkimuksen tarkoitus kuitenkin harkittava ja tutkimus rajattava erittäin tarkkaan. Tutkimuksen monimutkaisuuden kasvaessa myös tutkittavien ilmiöiden ja mahdollisten testien määrä kasvaa räjähdysmäisesti.

8.2 Suositukset

Sovelluskehittäjien kannattaa huomioida, että MySQL käyttää oletusarvoisesti eristyvyystasoa REPEATABLE READ moniversioinnin ollessa tällä eristyvyystasolla oletusarvoisesti päällä. Transaktiossa kannattaa käyttää eristyvyystasoa SERIALIZABLE aina, kun siitä ei ole suorituskyvylle liikaa haittaa. Lisäksi käyttämällä SELECT…LOCK IN SHARE MODE ja SELECT…FOR UPDATE –lauseita alemmilla eristyvyystasoilla voidaan välttyä useilta ongelmilta.

Transaktion odottaessa lukon saamista lock timeoutiin asti on sovelluskehittäjien muistettava peruuttaa (ROLLBACK) ja käynnistää kyseinen transaktio uudelleen. InnoDB ei pakota tätä, vaan peruuttaa ainoastaan kyseisen timeoutiin johtaneen komennon. Tämä saattaa johtaa rikkonaiseen tilanteeseen tietokannassa, mikäli tämän jälkeen annetaan COMMIT-komento.

Sovelluskehittäjien on myös huomioitava, että oletusasetuksilla InnoDB sallii tietokannan rakennetta muuttavien lauseiden suorittamisen myös silloin, kun keskeneräiset transaktiot ovat

viitanneet niihin. Tämä saattaa aiheuttaa vakavia ongelmia, mikäli tietokannan rakenteen muuttamista ei suunnitella huolella.

Lähteet

Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil P. 1995. A Critique of ANSI SQL Isolation Levels. Luettavissa: http://research.microsoft.com/pubs/69541/tr-95-51.pdf. Luettu 20.8.2010.

Cabral, S., Murphy, K. 2009. MySQL Administrator's Bible. Wiley. Indianapolis, Yhdysvallat.

Date, S. 2003. An Introduction to Database Systems. 8. painos. Addison-Wesley. Boston, Yhdysvallat.

Gulutzan, P., Pelzer, T. 1999. SQL-99 Complete, Really. R&D Books. Kansas, Yhdysvallat.

Hovi, A. 2010. SQL-opas. 9. painos. WSOY. Jyväskylä, Suomi.

IBM 2010a. Luettavissa: http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/ad/c0011916.htm. Luettu 10.8.2010.

IBM 2010b. Luettavissa: http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.wn.doc/doc/c0053503.html. Luettu 12.11.2010.

Innobase 2010. Luettavissa: http://www.innodb.com/wp/wp-content/uploads/2009/05/innovative-technologies-final.pdf. Luettu 16.9.2010.

Laiho, M., Laux, F. 2010. On SQL Concurrency technologies. Luettavissa: http://www.dbtechnet.org/papers/SQL_ConcurrencyTechnologies.pdf. Luettu 12.11.2010.

Microsoft 2010a. Understanding Locking in SQL Server. Luettavissa: http://msdn.microsoft.com/en-us/library/aa213039%28SQL.80%29.aspx. Luettu: 14.8.2010.

Microsoft 2010b. Isolation Levels in the Database Engine. Luettavissa: http://msdn.microsoft.com/en-us/library/ms189122.aspx. Luettu: 23.7.2010.

MySQL 2008. Sun to Acquire MySQL. Luettavissa: http://www.mysql.com/news-and-events/sun-to-acquire-mysql.html. Luettu: 23.7.2010.

MySQL 2010a. History of MySQL. Luettavissa: http://dev.mysql.com/doc/refman/5.1/en/history.html . Luettu: 23.7.2010.

MySQL 2010b. MySQL Licencing Policy. Luettavissa: http://www.mysql.com/about/legal/licensing/index.html. Luettu 30.7.2010

MySQL 2010c. The MyISAM Storage Engine. Luettavissa: http://dev.mysql.com/doc/refman/5.5/en/myisam-storage-engine.html. Luettu 24.7.2010.

MySQL 2010d. Setting the Storage Engine. Luettavissa: http://mysql2.mirrors-r-us.net/doc/refman/5.1/en/storage-engine-setting.htm. Luettu 24.7.2010

MySQL 2010e. MySQL Workbench Introduction. Luettavissa: http://dev.mysql.com/doc/workbench/en/wb-intro.html. Luettu 17.8.2010

MySQL 2010f. SQL Development. Luettavissa: http://dev.mysql.com/doc/workbench/en/wb-sql-development.html. Luettu 19.8.2010

MySQL 2010g. Data Modeling. Luettavissa: http://dev.mysql.com/doc/workbench/en/wb-data-modeling.html. Luettu 19.8.2010

MySQL 2010h. Server Administration. Luettavissa: http://dev.mysql.com/doc/workbench/en/wb-server-administration-server-instance.html. Luettu 19.8.2010

MySQL 2010i. The InnoDB Transaction Model and Locking. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-transaction-model.html. Luettu 30.7.2010

MySQL 2010j. InnoDB Lock Modes. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-lock-modes.html. Luettu 19.8.2010

MySQL 2010k. InnoDB Table and Index Structures. Luettavissa: http://mysql2.mirrors-r-us.net/doc/refman/5.1/en/innodb-table-and-index.html. Luettu: 23.7.2010.

MySQL 2010l. InnoDB Multi-Versioning. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-multi-versioning.html. Luettu 20.8.2010

MySQL 2010m. Consistent Nonlocking Reads. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-consistent-read.html. Luettu 19.8.2010

MySQL 2010n. SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-locking-reads.html. Luettu 19.8.2010

MySQL 2010o. Implicit Transaction Commit and Rollback. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-implicit-commit.html. Luettu 30.7.2010

MySQL 2010p. InnoDB Startup Options and System Variables. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-parameters.html#sysvar_innodb_lock_wait_timeout. Luettu 5.11.2010.

MySQL 2010q. Deadlock Detection and Rollback. Luettavissa: http://dev.mysql.com/doc/refman/5.0/en/innodb-deadlock-detection.html. Luettu 12.11.2010.

Oracle 2009. Oracle Buys Sun. Luettavissa: http://www.oracle.com/us/corporate/press/018363. Luettu: 23.7.2010.

Oracle 2010. Luettavissa: http://wiki.oracle.com/page/InnoDB. Luettu 16.9.2010

Schneider, R., Gibson, D. 2008. Microsoft SQL Server 2008 All-in-One Desk Reference for Dummies. Wiley. Indianapolis, Yhdysvallat.

Wikipedia 2010a. MySQL. Luettavissa: http://en.wikipedia.org/wiki/Mysql. Luettu: 23.7.2010.

Wikipedia 2010b. MySQL AB. Luettavissa: http://en.wikipedia.org/wiki/MySQL_AB . Luettu: 23.7.2010.

Wikipedia 2010c. InnoDB. Luettavissa: http://en.wikipedia.org/wiki/InnoDB
Luettu: 15.8.2010

Wikipedia 2010d. SQLyog. Luettavissa: http://en.wikipedia.org/wiki/SQLyog. Luettu: 15.8.2010

Wikipedia 2010e. MySQL Workbench. Luettavissa:
http://en.wikipedia.org/wiki/MySQL_Workbench. Luettu: 15.8.2010

Wikipedia 2010f. Concurrency control. Luettavissa:
http://en.wikipedia.org/wiki/Concurrency_control. Luettu: 3.8.2010

Wikipedia 2010g. Timestamp-based concurrency control. Luettavissa:
http://en.wikipedia.org/wiki/Timestamp-based_concurrency_control. Luettu: 5.8.2010

Wikipedia 2010h. Optimistic concurrency control. Luettavissa:
http://en.wikipedia.org/wiki/Optimistic_concurrency_control. Luettu: 6.8.2010

Wikipedia 2010i. Oracle Database. Luettavissa:
http://en.wikipedia.org/wiki/Oracle_Database. Luettu: 10.8.2010

Wikipedia 2010j. IBM DB2. Luettavissa: http://en.wikipedia.org/wiki/IBM_DB2. Luettu: 10.8.2010

Wikipedia 2010k. Microsoft SQL Server. Luettavissa:
http://en.wikipedia.org/wiki/Microsoft_SQL_Server. Luettu: 10.8.2010

Wikipedia 2010l. Index (database). Luettavissa:
http://en.wikipedia.org/wiki/Index_(database)#Clustered

Liitteet

Liite 1. Ongelmaskenaario: Hukattu päivitys (Lost Update)

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja

id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili

id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario	Alaskenaario 1	Alaskenaario 2
	READ UNCOMMITTED	☑	☑	☒
	READ COMMITTED	☒	☑*	☒
	REPEATABLE READ	☒	☑*	☒
	SERIALIZABLE	☑	☑*	☑
☑	= Testattu			
☑*	= Testattu, kuvaus vain tekstinä.			
☒	= Ei testattu/itsestäänselvä			

Huomioita skenaarioista: Pääskenaarion kuvaus seuraavalla sivulla. Alaskenaario 1 on muuten samanlainen kuin pääskenaario, mutta siinä transaktio A commitoidaan transaktion B vielä odottaessa jolloin sen asettamat lukot vapautuvat (ks. alaskenaario 1). Alaskenaario on johdettu pääskenaariosta eristyvyystasolla SERIALIZABLE. Siinä pyritään luomaan lukkiuma laittamalla transaktiot A ja B molemmat odottamaan vastakkaiselta transaktiolta vapautuvaa lukkoa (ks. alaskenaario 2).

Pääskenaarion kuvaus:

use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @tili_id = 3001;
SET @saldo = 0;
SET @uusi_saldo = 0;

Vaihe	Transaktio A	Transaktio B
1	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;**	
2		**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;**
3	**SET @uusi_saldo = @saldo + 99;**	
4		**SET @uusi_saldo = @saldo + 88;**
5	**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;**	
		UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;

7	**COMMIT;**	
8		**COMMIT;**

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario)

```
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET @tili_id = 3001;
SET @saldo = 0;
SET @uusi_saldo = 0;
```

Vaihe	Transaktio A	Tilin 3001 saldo	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
		100		
1	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.03 sec)	100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		100	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.03 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SET @uusi_saldo = @saldo + 99;** Query OK, 0 rows affected (0.00 sec)	100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4		100	**SET @uusi_saldo = @saldo + 88;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A**: ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 100, Uncommitted: 199		
6			**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A**: X-lukko tiliin 3001. **Transaktio B**: Pyydetty X-lukko tiliin 3001 odottaa transaktion A päättymistä lock timeoutiin asti, tämän jälkeen ei lukkoja.
7			**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A**: ei näkyviä lukkoja.

78	(transaktio B timeoutin jälkeen) **COMMIT;** Query OK, 0 rows affected (0.00 sec)			
		199		

Huomioita: Hukattu päivitys ei onnistu edes alimmalla eristyvyystasolla. Tämä käytös on SQL-standardin mukaista (ks. luku Eristyvyystasot). Transaktio B:tä käyttävän sovelluksen oletetaan reagoivan timeoutiin joten transaktio B päätetään tässä ROLLBACK-komentoon.

ISOLATION LEVEL: READ UNCOMMITTED (alaskenaario 1).

```
/* Commitoidaan transaktio A ennen transaktion B timeoutia. */
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET @tili_id = 3001;
SET @saldo = 0;
SET @uusi_saldo = 0;
```

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyselyn/kyselyiden suorittamisen jälkeen.**
		100		
1	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 0 rows affected (0.00 sec)	100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		100	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SET @uusi_saldo = @saldo + 99;** Query OK, 0 rows affected (0.00 sec)	100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4		100	**SET @uusi_saldo = @saldo + 88;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0	Committed: 100; Uncommitted: 199		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
6		Committed: 100; Uncommitted: 199 (transaktio A), 188 (transaktio B)	**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;**	**Transaktio A:** X-lukko tiliin 3001 **Transaktio B**: Jää odottamaan X-lukkoa tiliin 3001 kunnes transaktio A antaa COMMIT-komennon.
7	**COMMIT;**			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 199 Uncommitted: 188 (transaktio B)		**Transaktio B:** ei näkyviä lukkoja.
		Committed: 199 Uncommitted: 188 (transaktio B)	(transaktion A COMMIT-kommennon jälkeen) Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warn-	**Transaktio B:** ei näkyviä lukkoja.

			ings: 0	
8			**COMMIT;**	
		188		

Huomioita: Päällekirjoitus onnistuu. Transaktion B odottaessa transaktio A:lta vapautuvaa lukkoa vaiheessa 6 annetaan transaktion A COMMIT-komento. Tämän jälkeen transaktio B suoriutuu loppuun jättäen tietokantaan väärän saldon tilille 3001 (tulisi olla 287). Sama ilmiö toistuu eristyvyystasoilla READ COMMITTED ja REPEATABLE READ. Vasta tiukin eristyvyystaso SERIALIZABLE poistaa ongelman estämällä vaiheen 5 UPDATE-komennon.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario)

```
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET @tili_id = 3001;
SET @saldo = 0;
SET @uusi_saldo = 0;
```

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyselyn/kyselyiden suorittamisen jälkeen.**
		100		
1	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 0 rows affected (0.00 sec)	100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		100	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SET @uusi_saldo = @saldo + 99;** Query OK, 0 rows affected (0.00 sec)	100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4		100	**SET @uusi_saldo = @saldo + 88;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	100		**Transaktio A:** Pyydetty X-lukko tiliin 3001, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. **Transaktio B**: S-lukko tiliin 3001.
		100		
6	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)			**Transaktio B:** ei näkyviä lukkoja.
7			**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warn-	**Transaktio B:** ei näkyviä lukkoja.

			ings: 0	
8			**COMMIT;** Query OK, 0 rows affected (0.08 sec)	
		188		

Huomioita: Eristyvyystasolla SERIALIZABLE transaktio A ei pysty suorittamaan vaiheen 5 UPDATE-komentoa, sillä transaktion B vaiheessa 2 muuttujaan hakema tieto muuttuisi vanhaksi. Timeoutin seuratessa sovelluksen oletetaan reagoivan tilanteeseen peruuttamalla (ROLLBACK) transaktio A:n. Tämän jälkeen transaktio B pääsee suoriutumaan loppuun ja commitoimaan tietokantaan arvon 188. Tämä ei riko tietokannan eheyttä.

ISOLATION LEVEL: SERIALIZABLE (alaskenaario 2)

-- Toimitaan samalla tavalla kuin pääskenaariossa, mutta annetaan vaiheen 5 jälkeen transaktion A odottaessa lukkoa myös transaktion B UPDATE-komento ja yritetään näin aiheuttaa lukkiuma (deadlock).

```
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET @tili_id = 3001;
SET @saldo = 0;
SET @uusi_saldo = 0;
```

Vaihe	Transaktio A	Tilin 3001 saldo	Transaktio B	Lukot kyselyn/kyselyiden suorittamisen jälkeen.
		100		
1	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	100		**Transaktio A: ei näkyviä lukkoja.** **Transaktio B: ei näkyviä lukkoja.**
2		100	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A: ei näkyviä lukkoja.** **Transaktio B: ei näkyviä lukkoja.**
3	**SET @uusi_saldo = @saldo + 99;** Query OK, 0 rows affected (0.00 sec)	100		**Transaktio A: ei näkyviä lukkoja.** **Transaktio B: ei näkyviä lukkoja.**
4		100	**SET @uusi_saldo = @saldo + 88;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A: ei näkyviä lukkoja.** **Transaktio B: ei näkyviä lukkoja.**

5	**UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** (jää odottamaan)			**Transaktio A:** Pyydetty X-lukko tiliin 3001, jota transaktio jää odottamaan. **Transaktio B**: S-lukko tiliin 3001.
6			(transaktion A odottaessa lukkoa) **UPDATE Tili SET saldo = @uusi_saldo WHERE id = @tili_id;** (seuraa lukkiuma (deadlock)) ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction	
	(Transaktio B valitaan uhriksi ja se keskeytyy. Tämän jälkeen transaktio A:n UPDATE-komento suoriutuu loppuun) Query OK, 1 row affected (9.88 sec) Rows matched: 1 Changed: 1 Warn-	Committed: 100 Uncommitted: 199		

	ings: 0			
7	**COMMIT;** Query OK, 0 rows affected (0.00 sec)			
		199		

Huomioita: Vaiheen 6 jälkeen kumpikin transaktio odottaa toiselta vapautuvaa lukkoa, joka johtaa lukkiumaan (deadlock). InnoDB reagoi tilanteeseen valitsemalla transaktion B uhriksi. Tämän jälkeen transaktio A pääsee suoriutumaan loppuun ja commitoi tietokantaan saldon 199. Tämä ei riko tietokannan eheyttä.

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja

id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili

id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario	Alaskenaario 1
	READ UNCOMMITTED	☑	☒
	READ COMMITTED	☑	☑
	REPEATABLE READ	☑	☑
	SERIALIZABLE	☑	☑
☑	= Testattu		
☒	= Ei testattu/itsestäänselvä		

Huomioita skenaarioista: Pääskenaarion kuvaus seuraavalla sivulla. Alaskenaario 1 on muuten samanalainen kuin pääskenaario, mutta siinä transaktio A antaakin vaiheessa 3 ROLLBACK-komennon sijaan COMMIT-komennon.

Pääskenaarion kuvaus:

use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @tili_id = 3001;
SET @saldo = 0;

Vaihe	Transaktio A	Transaktio B
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;**	
2		**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;**
3	**ROLLBACK;**	
4		**SELECT @saldo AS 'tilin 3001 saldo';**

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario)

USE Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SET @tili_id = 3001;

SET @saldo = 0;

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyse-lyn/kyselyiden suo-rittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 200, Uncommitted: 100		
2		Committed: 200, Uncommitted: 100	**SELECT saldo IN-TO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
3	**ROLLBACK;** Query OK, 0 rows affected (0.17 sec)			**Transaktio B**: ei näkyviä lukkoja.
		100		

4		100	**SELECT @saldo AS 'tilin 3001 saldo';** +----------+ | tilin 3001 saldo | +----------+ | 200 | +----------+	**Transaktio B** : ei näkyviä lukkoja.
		100	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	

Huomioita:

Likainen luku (dirty read) onnistuu, sillä muuttujan @saldo arvo kuvaa tilannetta, joka ei pidä enää paikkaansa tietokannassa. Tämä käytös on SQL-standardin mukaista.

ISOLATION LEVEL: READ COMMITTED (pääskenaario)

use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET @tili_id = 3001;
SET @saldo = 0;

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyse-lyn/kyselyiden suo-rittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 200, Uncommitted: 100		
2		Committed: 200, Uncommitted: 100	**SELECT saldo IN-TO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
3	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)			**Transaktio B**: ei näkyviä lukkoja.

4		100	**SELECT @saldo AS 'tilin 3001 saldo';** +----------+ | tilin 3001 saldo | +----------+ | 100 | +----------+	**Transaktio B**: ei näkyviä lukkoja.
		100	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	
		100		

Huomioita:

READ UNCOMMITTED –tasolla esiintynyt likainen luku -ongelma poistui, mutta samalla syntyi uusi ongelma. Jos transaktio A antaakin nyt COMMIT-komennon, transaktiolla B on tästä eteenpäin käytössä väärää tietoa (ks. alaskenaario 1) Ongelma toistuu myös eristyvyystasolla REPEATABLE READ. Vasta eristyvyystaso SERIALIZABLE ratkaisee ongelman estämällä transaktion A vaiheessa 2 tekemän SELECT-lauseen ennen transaktion B päättymistä.

ISOLATION LEVEL: READ COMMITTED (alaskenaario 1)

/* Toimitaan muuten samoin kuin pääskenaariossa, mutta annetaan vaiheessa 3 ROLLBACK-kommennon sijaan COMMIT jolloin muuttujaan haettu tieto muuttuisi vääräksi. */
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET @tili_id = 3001;
SET @saldo = 0;

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyse-lyn/kyselyiden suo-rittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 200, Uncommitted: 100		
2		Committed: 200, Uncommitted: 100	**SELECT saldo IN-TO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
3	**COMMIT;**			
4		200	**SELECT @saldo AS 'tilin 3001 saldo';**	**Transaktio B**: ei näkyviä lukkoja.

			+----------+ | tilin 3001 saldo | +----------+ | 100 | +----------+	
		200	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	
		200		

Huomioita: Transaktio B hakee muuttujaan uusimman commitoidun tiedon vaikka siihen on tehty ei-commitoituja muutoksia. Transaktio B:n annettua COMMIT-komennon transaktio A:n muuttujaan hakema tieto muuttuu vanhaksi.

ISOLATION LEVEL: REPEATABLE READ (alaskenaario 1)

/* Toimitaan muuten samoin kuin pääskenaariossa, mutta annetaan vaiheessa 3 ROLLBACK-kommennon sijaan COMMIT jolloin muuttujaan haettu tieto muuttuisi vääräksi. */

use Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET @tili_id = 3001;

SET @saldo = 0;

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyse-lyn/kyselyiden suo-rittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 200, Uncommitted: 100		
2		Committed: 200, Uncommitted: 100	**SELECT saldo IN-TO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
3	**COMMIT;**			
4		200	**SELECT @saldo AS 'tilin 3001 saldo';**	**Transaktio B**: ei näkyviä lukkoja.

			+----------+ | tilin 3001 saldo | +----------+ | 100 | +----------+	
		200	**SELECT @saldo AS 'tilin 3001 saldo';** Query OK, 0 rows affected (0.00 sec)	
		200		

Huomioita: Testin tulos on sama kuin eristyvyystasolla READ COMMITTED. Transaktio B hakee muuttujaan uusimman commitoidun tiedon vaikka siihen on tehty ei-commitoituja muutoksia. Transaktio B:n annettua COMMIT-komennon transaktio A:n muuttujaan hakema tieto muuttuu vanhaksi.

ISOLATION LEVEL: SERIALIZABLE (alaskenaario 1)

use Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET @tili_id = 3001;

SET @saldo = 0;

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyselyn/kyselyiden suorittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 200, Uncommitted: 100		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
2		Committed: 200, Uncommitted: 100 80	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** S-lukko tiliin 3001; **Transaktio B**: Pyydetty X-lukko tiliin 3001, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.
3	**COMMIT;**			**Transaktio B**: ei

				näkyviä lukkoja.
4		200		

Huomioita: Ongelma poistuu SERIALIZABLE-eristyvyystasolla. Transaktio B ei pysty hakemaan tietoa muuttujaan vaiheessa 2 johtuen tilille 3001 asetetusta X-lukosta.

ISOLATION LEVEL: REPEATABLE READ (pääskenaario)

use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET @tili_id = 3001;
SET @saldo = 0;

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyselyn/kyselyiden suorittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.07 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 100 Uncommitted: 200		
2		Committed: 100 Uncommitted: 200	**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
3	**ROLLBACK;**			**Transaktio B**: ei näkyviä lukkoja.

	Query OK, 0 rows affected (0.04 sec)			
		100		
4			**SELECT @saldo AS 'tilin 3001 saldo';** +------------------+ | tilin 3001 saldo | +------------------+ | 100 | +------------------+	**Transaktio B**: ei näkyviä lukkoja.
5			**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	
		100		

Huomioita:

Testin tulos on sama kuin READ COMMITTED –eristyvyystasolla. Likainen luku –ongelmaa ei esiinny.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario)

```
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET @tili_id = 3001;
SET @saldo = 0;
```

Vaihe	**Transaktio A**	**Tilin 3001 saldo**	**Transaktio B**	**Lukot kyselyn/kyselyiden suorittamisen jälkeen.**
		100		
1	**UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		Committed: 100 Uncommitted: 200		
2			**SELECT saldo INTO @saldo FROM Tili WHERE id = @tili_id;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** S-lukko tiliin 3001; **Transaktio B**: Pyydetty X-lukko tiliin 3001, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.
3	**ROLLBACK;** Query OK, 0 rows affected (0.18 sec)			**Transaktio B**: ei näkyviä lukkoja.
		100		

4		100	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	

Huomioita:

Testi antaa eri tuloksen kuin READ COMMITTED ja REPEATABLE READ –eristyvyystasoilla. Transaktio B ei voi kohdistaa SELECT-lausetta sellaiseen riviin, johon on tehty ei-committoituja muutoksia. Lisäksi huomioitavaa, että mikäli kohdan 2 jälkeen transaktio A antaa ROLLBACK-komennon sillä aikaa kun transaktio B odottaa lukon saamista, suoriutuu transaktio B loppuun asti ja tulostaa oikean saldon 100.

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja		
id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili		
id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario
	READ UNCOMMITTED	☒
	READ COMMITTED	☑
	REPEATABLE READ	☑
	SERIALIZABLE	☑
☑	= Testattu	
☒	= Ei testattu/itsestäänselvä	

Pääskenaarion kuvaus:

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

Vaihe	Transaktio A	Transaktio B
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;**	
2		**UPDATE Tili SET saldo = saldo + 77 WHERE id = 2001;**
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;**	
4		**COMMIT;**
5	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;**	

Huomioita pääskenaariosta:

Transaktio A:n suorittaman SELECT-lauseen tulisi antaa joka kerta sama tulos. Jos näin ei ole, haku ei ole toistettavissa ja kyseessä on muuttuva lukujoukko.

ISOLATION LEVEL: READ COMMITTED (pääskenaario)

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

Vaihe	Transaktio A	Keijon tilien 2001, 2002 ja 2003 yhteissaldo	Transaktio B	Lukot kyse-lyn/kyselyiden suorit-tamisen jälkeen.
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1200	1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2			**UPDATE Tili SET saldo = saldo + 77 WHERE id = 2001;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 1200 Uncommitted: 1277		
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1200	Committed: 1200 Uncommitted: 1277		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4			**COMMIT;** Query OK, 0 rows affected	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä

			(0.01 sec)	lukkoja.
5	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1277	1277		
	ROLLBACK; Query OK, 0 rows affected (0.01 sec)			
		1277		

Huomioita: Transaktion A tekemä SELECT-lause ei anna samaa tulosta jokaisella kerralla, joten kyseessä on muuttuva lukujoukko. READ COMMITTED –eristyvyystasolla luetaan aina uusin commitoitu tieto, mutta tämän muuttumista ei estetä millään tavalla.

ISOLATION LEVEL: REPEATABLE READ (pääskenaario)

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Vaihe	Transaktio A	Keijon tilien 2001, 2002 ja 2003 yhteissaldo	Transaktio B	Lukot kyse-lyn/kyselyiden suorit-tamisen jälkeen.
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1200	1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2			**UPDATE Tili SET saldo = saldo + 77 WHERE id = 2001;** Query OK, 1 row affected (0.06 sec) Rows matched: 1 Changed: 1 Warnings: 0	**Transaktio A**: ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 1200 Uncommitted: 1277		
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1200	Committed: 1200 Uncommitted: 1277		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4			**COMMIT;** Query OK, 0 rows affected (0.01 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1200	1277		
	ROLLBACK; Query OK, 0 rows affected (0.01 sec)			
		1277		

Huomioita: Transaktion A tekemä SELECT-lause antaa saman tuloksen jokaisella kerralla, joten lukuoperaatio on toistettavissa (repeatable). Muuttuvan/pienenevän lukujoukon ongelmaa ei esiinny, mutta transaktion A ”näkemä” tieto vanhenee, kun transaktio B muuttaa kyseistä riviä. InnoDB:n käyttämä moniversiointi perustuu REPEATABLE READ –eristyvyystasolla siihen, että transaktion tekemä SELECT-lause palauttaa joka kerta sen tuloksen, minkä se sai tehtyään tämän SELECT-lauseen ensimmäisen kerran. Tämän tiedon muuttumista ei estetä millään tavalla.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario)

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Vaihe	Transaktio A	Keijon tilien 2001, 2002 ja 2003 yhteissaldo	Transaktio B	Lukot kyselyn/kyselyiden suorittamisen jälkeen.
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id=2;** -> 1200	1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2			**UPDATE Tili SET saldo = saldo + 77 WHERE id = 2001;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** S-lukko tiliin 2001. **Transaktio B**: Pyydetty X-lukko tiliin 2001, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.
			ROLLBACK; Query OK, 0 rows affected (0.00 sec)	
	ROLLBACK; Query OK, 0 rows affected (0.00 sec)			
		1200		

Huomioita: Transaktio B ei onnistu päivittämään riviä, josta transaktio A on hakenut tietoa SELECT-lauseella. Näin estetään sekä se, että transaktion A tekemä SELECT-lause ei olisi toistettavissa samalla tuloksella sekä se, että kyseisen SELECT-lauseen tuottama tulos vanhentuisi. Testiä ei ole syytä jatkaa pitemmälle, joten molemmat transaktiot päätetään tässä ROLLBACK-komentoon.

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja		
id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili		
id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario	Alaskenaario 1
	READ UNCOMMITTED	☒	☒
	READ COMMITTED	☒	☒
	REPEATABLE READ	☑	☒
	SERIALIZABLE	☑	☑
☑	= Testattu		
☒	= Ei testattu/itsestäänselvä		

Pääskenaarion kuvaus:

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

Vaihe	Transaktio A	Transaktio B
1	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;**	
2		**INSERT INTO Tili VALUES (2004,66,2);**
3	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;**	
4		**COMMIT;**
5	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;**	

Huomioita skenaarioista:

Pääskenaariossa Transaktio A laskee Keijon tilien lukumäärän. Tämän jälkeen transaktio B luo Keijolle uuden tilin, jolloin Keijolla on entisen kolmen tilin sijaan nyt neljä tiliä. Mikäli transaktio A ei lukuoperaation toistaessaan ”näe” kyseistä tiliä, on kyseessä kasvava lukujoukko. Uutta tiliä kutsutaan tässä yhteydessä haamuriviksi (phantom row).

Alaskenaariossa 1 tutkitaan pääskenaariossa ilmenneen indeksivälilukon toimintaa laajennetulla aineistolla . InnoDB:n indeksivälilukosta tarjoamat tiedot ovat vajavaiset, joten testin avulla yritetään selvittää, minkälaisia uusia rivejä tauluun on mahdollista lisätä.

ISOLATION LEVEL: REPEATABLE READ (pääskenaario)

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Vaihe	Transaktio A	Keijon tilien lukumäärä	Transaktio B	Lukot kyselyn/kyselyiden suorittamisen jälkeen.
1	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;** -> 3	3		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
2			**INSERT INTO Tili VALUES (2004,66,2);** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 3 Uncommitted: 4		
3	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;** -> 3	Committed: 3 Uncommitted: 4		**Transaktio A**: ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4			**COMMIT;**	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;** -> 3 Huom. transaktio ei näe juuri luotua tiliä.	4		**Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;**			
		4		

Huomioita:

Transaktio A ei näe transaktion B luomaa uutta tiliä, joten kyseessä on kasvava lukujoukko. Transaktio A:n SELECT-lause palauttaa joka kerta saman tuloksen, joten haku on toistettavissa (Repeatable), mutta haamurivin syntymistä ei estetä. Toiminta on SQL-standardin mukaista.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario)

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Vaihe	Transaktio A	Keijon tilien lukumäärä	Transaktio B	Lukot kyselyn/kyselyiden suorittamisen jälkeen.
1	**SELECT COUNT(id) AS 'Keijon tilien lukumäärä' FROM Tili WHERE omi_id = 2;** -> 3	3		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
2			**INSERT INTO Tili VALUES (2004,66,2);** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** Indeksivälilukko (S, GAP). InnoDB ei anna tarkempia tietoja. **Transaktio B**: Pyydetty indeksivälilukko (X,GAP) jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. InnoDB ei anna tarkempia tietoja.
		3		
		3	**ROLLBACK;** Query OK, 0 rows affected (0.01 sec)	**Transaktio A:** ei näkyviä lukkoja.
	ROLLBACK; Query OK, 0 rows affected (0.01 sec)	3		

Huomioita:

Haamurivin lisääminen ei onnistu, sillä transaktio A:n indeksivälilukko (S, GAP) estää transaktio B:tä lisäämästä Keijolle uutta tiliä. Toimina on SQL-standardin mukaista. InnoDB ei anna asetetusta indeksivälilukosta yksityiskohtaisempaa tietoa.

ISOLATION LEVEL: SERIALIZABLE (alaskenaario 1)

/* Tutkitaan pääskenaariossa ilmenneen indeksivälilukon toimintaa laajennetulla aineistolla . InnoDB:n indeksivälilukosta tarjoamat tiedot ovat vajavaiset, joten testin avulla yritetään selvittää, minkälaisia uusia rivejä tauluun on mahdollista lisätä kun Mister X:n tileistä (omi_id = 4) on laskettu lukymäärä COUNT()-funtiolla. */

id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen
4	Mister	X
5	Viivi	Vitonen

id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3
3002	99	3
4001	88	4
4002	77	4
5001	66	5
5002	55	5

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB
```

```
CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB
```

```
ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES
(2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen'),(4,'Mister','X'),(5,'¨Viivi','Vitonen');

INSERT INTO `tili` VALUES
(2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3),(3002,99,3),(4001,88,4),(4002,77,4),(5001,6
6,5),(5002,55,5);
```

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn/kyselyiden suorittamisen jälkeen.**
1	**SELECT COUNT(id) AS 'Mister X:n tilien lukumäärä' FROM Tili WHERE omi_id = 4;** -> 2		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**INSERT INTO Tili VALUES (9999,44,2);** Query OK, 1 row affected (0.00 sec) **ROLLBACK;** Query OK, 0 rows affected (0.05 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3		**INSERT INTO Tili VALUES (9999,44,3);**	**Transaktio A:** S-lukko tiliin 4001. **Transaktio B:** Pyydetty

		ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	indeksivälilukko (X,GAP) jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.
4		**INSERT INTO Tili VALUES (9999,44,4);** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** Indeksivälilukko (S,GAP). InnoDB ei anna tarkempia tietoja. **Transaktio B:** Pyydetty indeksivälilukko (X,GAP) jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. InnoDB ei anna tarkempia tietoja.
5		**INSERT INTO Tili VALUES (9999,44,5);** Query OK, 1 row affected (0.00 sec) **ROLLBACK;** Query OK, 0 rows affected (0.05 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
6		**INSERT INTO Tili VALUES (3000,44,2);** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

		ROLLBACK; Query OK, 0 rows affected (0.05 sec)	
7		**INSERT INTO Tili VALUES (3333,44,2);** Query OK, 1 row affected (0.00 sec) **ROLLBACK;** Query OK, 0 rows affected (0.05 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		INSERT INTO Tili VALUES (3000,44,5); ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** Indeksivälilukko (S,GAP). InnoDB ei anna tarkempia tietoja. **Transaktio B:** Pyydetty indeksivälilukko (X,GAP) jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. InnoDB ei anna tarkempia tietoja.
8		**INSERT INTO Tili VALUES (3333,44,5);** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transac-	**Transaktio A:** Indeksivälilukko (S,GAP). InnoDB ei anna tarkempia tietoja. **Transaktio B:** Pyydetty indeksivälilukko (X,GAP) jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. InnoDB ei

		tion	anna tarkempia tietoja.
		INSERT INTO Tili VALUES (1999,44,2); Query OK, 1 row affected (0.00 sec) **ROLLBACK;** Query OK, 0 rows affected (0.01 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		INSERT INTO Tili VALUES (4999,44,5); ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** Indeksiväli-lukko (S,GAP). InnoDB ei anna tarkempia tietoja. **Transaktio B:** Pyydetty indeksivälilukko (X,GAP) jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. InnoDB ei anna tarkempia tietoja.
		ROLLBACK; Query OK, 0 rows affected (0.01 sec)	**Transaktio A:** ei näkyviä lukkoja.
	ROLLBACK; Query OK, 0 rows affected (0.01 sec)		

Huomioita:

id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen
4	Mister	X
5	Viivi	Vitonen

id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3
3002	99	3
4001	88	4
4002	77	4
5001	66	5
5002	55	5

Huomioita:

Tili-taulusta on asetettu indeksivälilukko (S, GAP) omi_id:lle 4, josta transaktio A ensin laski lukumäärän COUNT()-funtiolla. Merkillepantavaa on se, että tämän lisäksi on asetettu S-lukot niille riveille, joissa omi_id on 3. Toisin sanoen omistajille 2 ja 5 on mahdollista lisätä uusi tili, mutta ei omistajille 3 tai 4, vaikka vain omistajan 4 tileistä on laskettu lukumäärä.

Toinen mielenkiintoinen havainto on se, että omistajalle 5 on mahdollista lisätä ainoastaan sellainen rivi, jossa id:n arvo on suurempi kuin olemassaolevien rivien id:t. Toisin sanoen esimerkisi lause **INSERT INTO Tili VALUES (9999,44,5);** onnistuu, mutta lause **INSERT INTO Tili VALUES (4999,44,5);** epäonnistuu. Omistajalle 2 on taas mahdollista lisätä myös sellainen rivi, jossa id on pienempi kuin omistajan 2 tilien pienin id 2001. Tämä johtuu mahdollisesti InnoDB:n käyttäymistä ryvästetyistä indekseistä (Clustered Indexes, ks. luku InnoDB:n toteutus).

Testiä varten luotavat taulut:

```
CREATE TABLE `Testi` (
  `id` int(10) unsigned NOT NULL,
  `s` varchar(20),
  `n` int default 0,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

INSERT INTO `testi` VALUES (1,'tekstiä ',0), (2,'tekstiä ',0);
```

Taulun alkuarvot ovat seuraavat:

Testi

id	s	n
1	tekstiä	0
2	tekstiä	0

Testatut eristyvyystasot:

		Pääskenaario	Alaskenaario 1
	READ UNCOMMITTED	☑	☑
	READ COMMITTED	☑	☑
	REPEATABLE READ	☑	☑
	SERIALIZABLE	☑	☑
☑	= Testattu		
☒	= Ei testattu/itsestäänselvä		

Pääskenaarion kuvaus:

USE Testi;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

Vaihe	Transaktio A	Transaktio B
1	**Update TESTI SET s='uusi arvo 1', n=n+1 WHERE id = 1;**	
2		**Update TESTI SET s='uusi arvo 2', n=n+2 WHERE id = 2;**
3	**Update TESTI SET s='uusi arvo 3', n=n+1 WHERE id = 2;**	
4		**Update TESTI SET s='uusi arvo 4', n=n+2 WHERE id = 1;**
5		**COMMIT;**
6	**COMMIT;**	

Huomioita skenaarioista:

Pääskenaariossa transaktiot A ja B päivittävät kahta (tässä tapauksessa saman taulun) riviä riviä eri järjestyksessä. Transaktio A päivittää riviä 1 ensimmäisenä, Transaktio B taas päivittää ensimmäisenä riviä 2. Tämän jälkeen molemmat transaktiot yritetään commitoida. Mikäli transaktio A ja B odottavat molemmat vastakkaiselta transaktiolta vapautuvaa lukkoa, kyseessä on lukkiumatilanne (deadlock) jonka tietokannanhallintajärjestelmän (MySQL:n tapauksessa tietokantamoottorin) täytyy ratkaista jollain tavalla. Lisäksi on tärkeää tietää, selviääkö jompi kumpi transaktioista commitoimaan päivittämänsä tiedot.

Alaskenaariossa 1 annetaan vaiheen 4 UPDATE-komento kun transaktio A odottaa vielä lukon saamista vaiheen 3 jälkeen. Tämän tulisi johtaa lukkiumaan (deadlock). Tarkoituksena on selvittää, miten InnoDB reagoi tilanteeseen.

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario).

```
/*tämä testi tuottaa saman tuloksen kaikilla eristyvyystasoilla/*
USE Testi;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

Vaihe	Transaktio A	Testi-taulun arvot (id, s, n)	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
		1 | tekstiä | 0 2 | tekstiä | 0		
1	**UPDATE Testi SET s='uusi arvo 1', n=n+1 WHERE id = 1;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 1 | tekstiä | 0 2 | tekstiä | 0 Uncommitted: 1 | uusi arvo 1 | 1 2 | tekstiä | 0		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2			**UPDATE Testi SET s='uusi arvo 2', n=n+2 WHERE id = 2;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

			Rows matched: 1 Changed: 1 Warnings: 0	
		Committed: 1 | tekstiä | 0 2 | tekstiä | 0 Uncommitted: 1 | uusi arvo 1 | 1 2 | uusi arvo 2 | 0		
3	**UPDATE Testi SET s='uusi arvo 3', n=n+1 WHERE id = 2;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction			**Transaktio A:** Pyydetty X-lukko riviin id = 2, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu. **Transaktio B**: X-lukko riviin id = 2.
4			**UPDATE Testi SET s='uusi arvo 4', n=n+2 WHERE id = 1;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** X-lukko riviin id = 1. **Transaktio B**: Pyydetty X-lukko riviin id = 1, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.

5			**COMMIT;**	**Transaktio B:** ei näkyviä lukkoja.
6	**COMMIT;**			
		1 | uusi arvo 1 | 1 2 | uusi arvo 2 | 2		

Huomioita:

Tämä testi tuottaa saman tuloksen kaikilla eristyvyystasoilla. Molemmat transaktiot epäonnistuvat sen rivin päivittämisessä, mihin toinen transaktio on tehnyt ei-commitoituja päivityksiä. Lock timeout ei kuitenkaan päätä kuitenkaan koko transaktiota ROLLBACK-komentoon, vaan ainoastaan kyseinen epäonnistunut UPDATE-lause perutaan. Kumpikin transaktio onnistuu commitoimaan tietokantaan ensimmäisenä päivittämänsä rivin uuden arvon.

ISOLATION LEVEL: READ UNCOMMITTED (alaskenaario 1).

```
/*Annetaan vaiheen 4 UPDATE-komento kun vaihe 3 odottaa vielä lukon vapautumista. Tämä testi tuottaa saman tuloksen kaikilla eristyvyystasoilla*/
USE Testi;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

Vaihe	Transaktio A	Testi-taulun arvot (id, s, n)	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
		1 | tekstiä | 0 2 | tekstiä | 0		
1	**UPDATE Testi SET s='uusi arvo 1', n=n+1 WHERE id = 1;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0			**Transaktio A**: ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
		Committed: 1 | tekstiä | 0 2 | tekstiä | 0 Uncommitted: 1 | uusi arvo 1 | 1 2 | tekstiä | 0		
2			**UPDATE Testi SET s='uusi arvo 2', n=n+2 WHERE id = 2;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0	**Transaktio A**: ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

		Committed: 1 | tekstiä | 0 2 | tekstiä | 0 Uncommitted: 1 | uusi arvo 1 | 1 2 | uusi arvo 2 | 0		
3	**UPDATE Testi SET s='uusi arvo 3', n=n+1 WHERE id = 2;** (jää odottamaan)			**Transaktio A:** Pyydetty X-lukko riviin id = 2, jota transaktio jää odottamaan. **Transaktio B**: X-lukko riviin id = 2.
4			(transaktion A odottaessa vielä lukon vapautumista) **UPDATE Testi SET s='uusi arvo 4', n=n+2 WHERE id = 1;** ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting trans action (transaktio B valitaan uhriksi	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

			jonka jälkeen transaktio A pääsee jatka-maan loppuun asti)	
		Committed: 1 | tekstiä | 0 2 | tekstiä | 0 Uncommitted: 1 | uusi arvo 1 | 1 2 | uusi arvo 3 | 1		.
	Query OK, 1 row affected (11.51 sec) Rows matched: 1 Changed: 1 Warnings: 0			
5			**COMMIT;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B**: ei näkyviä lukkoja.
		(transaktio B jättänyt mitään muutoksia tie-tokantaan) Committed: 1 | tekstiä | 0 2 | tekstiä | 0 Uncommitted: 1 | uusi arvo 1 | 1		

		2 | uusi arvo 3 | 1		
6	**COMMIT;**			**Transaktio A:** ei näkyviä lukkoja. **Transaktio B: e**i näkyviä lukkoja.
		1 | uusi arvo 1 | 1 2 | uusi arvo 3 | 1		

Huomioita:

Tämä testi tuottaa saman tuloksen kaikilla eristyvyystasoilla. Transaktio B valitaan lukkiumatilanteessa uhriksi. Tässä tilanteessa kaikki sen tekemät muutokset peruutetaan (ROLLBACK), sillä seuraava COMMIT-komento ei jätä mitään muutoksia tietokantaan. Transaktio A puolestaan voittaa molempien rivien, myös jälkimmäisenä päivittämänsä rivin päivityksen ja commitoi muutokset tietokantaan.

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja		
id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili		
id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario
	READ UNCOMMITTED	☑
	READ COMMITTED	☑
	REPEATABLE READ	☑
	SERIALIZABLE	☑
☑	= Testattu	
☒	= Ei testattu/itsestäänselvä	

Pääskenaarion kuvaus

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @omistaja_id = 2;

Vaihe	Transaktio A	Transaktio B
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;**	
2		**DELETE FROM Omistaja WHERE id = @omistaja_id;**
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;**	
4		**COMMIT;**
5	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;**	

Huomioita pääskenaariosta:

Tilin ja sen omistajan välille on määritelty viite-eheyssääntö ON DELETE CASCADE ON UPDATE CASCADE. Näin poistettaessa Omistaja-taulusta omistaja pitäisi kaikkien kyseisen omistajan tilien poistua Tilit-taulusta. Mikäli isärivin poistaminen onnistuu, ei transaktion A tekemä SELECT-lause ole enää toistettavissa sillä kyseisiä tilejä ei ole enää olemassa.

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario).

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET @omistaja_id = 2;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**DELETE FROM Omis-taja WHERE id = @omistaja_id;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> NULL		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

4		**COMMIT;** Query OK, 0 rows affected (0.05 sec)	**Transaktio B:** ei näkyviä lukkoja.
5	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> NULL		**Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Transaktio B onnistuu poistamaan isärivin ja tähän liittyvät lapsirivit, vaikka transaktio A on kohdistanut SELECT-lauseen kyseisiin riveihin. Tämän jälkeen transaktio A saa tuloksen NULL toistaessaan SELECT-lauseen vaiheessa 3.

ISOLATION LEVEL: READ COMMITTED (pääskenaario).

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET @omistaja_id = 2;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**DELETE FROM Omistaja WHERE id = @omistaja_id;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

4		**COMMIT;** Query OK, 0 rows affected (0.05 sec)	**Transaktio B:** ei näkyviä lukkoja.
5	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> NULL		**Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Transaktio B onnistuu poistamaan isärivin ja tähän liittyvät lapsirivit, vaikka transaktio A on kohdistanut SELECT-lauseen kyseisiin riveihin. Testin tulos eroaa eristyvyystasosta READ UNCOMMITTED siinä, että READ COMMITTED –eritysyystasolla (kuten saatetaan oletaa) transaktio A saa SELECT-lauseellee tuloksen NULL vasta siinä vaiheessa, kun transaktio B on commitoinut isärivin poistamisen tietokantaan.

ISOLATION LEVEL: REPEATABLE READ (pääskenaario).

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET @omistaja_id = 2;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**DELETE FROM Omis-taja WHERE id = @omistaja_id;** Query OK, 1 row affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

4		**COMMIT;**	**Transaktio B:** ei näkyviä lukkoja.
5	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Transaktio B onnistuu poistamaan isärivin ja tähän liittyvät lapsirivit, vaikka transaktio A on kohdistanut SELECT-lauseen kyseisiin riveihin. Testin tulos eroaa eristyvyystasosta READ COMMITTED siinä, että transaktio A:n SELECT-lause palauttaa edelleen saman tuloksen myös sen jälkeen kun transaktio B on commitoinut isärivin poistamisen tietokantaan.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario).

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET @omistaja_id = 2;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1200		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**DELETE FROM Omistaja WHERE id = @omistaja_id;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** S-lukko tiliin 2001. **Transaktio B**: Pyydetty X-lukko tiliin 2001, jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.
3	**SELECT SUM(saldo) FROM Tili WHERE omi_id =**		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei

	@omistaja_id; -> 1200		näkyviä lukkoja.
4		**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)	
5	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Transaktio B ei onnistu poistamaan tietokannasta rivejä, joihin transaktio A on kohdistanut SELECT-lauseen. Oletettavasti InnoDB on asettanut S-lukot kaikille omistajan 2 tileille, mutta näistä näytetään vain ensimmäinen (2001) johon konflikti kohdistuu. Testiä ei ole syytä jatkaa pitemmälle, joten molemmat transaktiot päätetään tässä ROLLBACK-komentoon.

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja		
id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili		
id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario
	READ UNCOMMITTED	☑
	READ COMMITTED	☑
	REPEATABLE READ	☑
	SERIALIZABLE	☑
☑	= Testattu	
☒	= Ei testattu/itsestäänselvä	

Pääskenaarion kuvaus:

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @omistaja_id = 2;

Vaihe	Transaktio A	Transaktio B
1	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;**	
2		**ALTER TABLE Omistaja DROP COLUMN enimi;**
3	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;**	
4		**COMMIT;**
5	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;**	

Huomioita pääskenaariosta:

Skenaarion alussa Transaktio A on hakenut tietoa SELECT-lauseella. Transaktion ollessa vielä käynnissä transaktio B yrittää muuttaa kyseisen taulun rakennetta pudottamalla sarakkeen, johon transaktio A viittaa. Transaktion A:n SELECT-lauseen hakema tieto on muuttunut vääräksi (saraketta ei ole enää olemassa) eikä kyseinen lause ole toistettavissa.

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario).

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET @omistaja_id = 2;
```

Vaihe	Transaktio A	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
1	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** -> Keijo, Kullervo		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**ALTER TABLE Omistaja DROP COLUMN enimi;** Query OK, 2 rows affected (0.18 sec) Records: 2 Duplicates: 0 Warnings: 0	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** ERROR 1054 (42S22): Unknown column 'enimi' in 'field list'		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4		**COMMIT;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** ERROR 1054 (42S22): Unknown column 'enimi' in 'field list'		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;** Query OK, 0 rows affected (0.04 sec)		

Huomioita:

Transaktio B onnistuu poistamaan sarakkeen enimi siitä huolimatta, että transaktio A on viitannut kyseiseen sarakkeeseen SELECT-lauseella. Transaktion A toistaessa kyseisen lauseen vaiheissa 3 ja 5 se epäonnistuu, sillä saraketta enimi ei ole enää olemassa.

ISOLATION LEVEL: READ COMMITTED (pääskenaario).

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET @omistaja_id = 2;
```

Vaihe	Transaktio A	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
1	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** -> Keijo, Kullervo		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**ALTER TABLE Omistaja DROP COLUMN enimi;** Query OK, 2 rows affected (0.20 sec) Records: 2 Duplicates: 0 Warnings: 0	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** ERROR 1054 (42S22): Unknown column 'enimi' in 'field list'		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4		**COMMIT;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** ERROR 1054 (42S22): Unknown column 'enimi' in 'field list'		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;** Query OK, 0 rows affected (0.04 sec)		

Huomioita:

Transaktio B onnistuu poistamaan sarakkeen enimi siitä huolimatta, että transaktio A on viitannut kyseiseen sarakkeeseen SELECT-lauseella. Transaktion A toistaessa kyseisen lauseen vaiheessa 5 se epäonnistuu, sillä saraketta enimi ei ole enää olemassa. Tulos eroaa eristyvyystasosta READ UNCOMMITTED (kuten olettaa saattaa) siinä, että transaktio A huomaa sarakkeen poistamisen vasta siinä vaiheessa, kun transaktio B on commitoinut kyseisen muutoksen tietokantaan.

ISOLATION LEVEL: REPEATABLE READ (pääskenaario).

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET @omistaja_id = 2;
```

Vaihe	Transaktio A	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
1	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** -> Keijo, Kullervo		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**ALTER TABLE Omistaja DROP COLUMN enimi;** Query OK, 2 rows affected (0.27 sec) Records: 2 Duplicates: 0 Warnings: 0	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** ERROR 1054 (42S22): Unknown column 'enimi' in 'field list'		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
4		**COMMIT;** Query OK, 0 rows affected (0.00 sec)	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.

5	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** ERROR 1054 (42S22): Unknown column 'enimi' in 'field list'		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
6	**ROLLBACK;** Query OK, 0 rows affected (0.04 sec)		

Huomioita:

Transaktio B onnistuu poistamaan sarakkeen enimi siitä huolimatta, että transaktio A on viitannut kyseiseen sarakkeeseen SELECT-lauseella. Poikkeavaa tämän testin tuloksissa on se, että SELECT-lauseen tulos ei ole toistettavissa (repeatable) eristyvyystason nimestä REPEATABLE READ huolimatta. Sarakkeen tai taulun poistaminen on ilmeisesti tässä mielessä erityistapaus.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario).

```
USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET @omistaja_id = 2;
```

Vaihe	Transaktio A	Transaktio B	Lukot kyselyn suorittamisen jälkeen.
1	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;** -> Keijo, Kullervo		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**ALTER TABLE Omistaja DROP COLUMN enimi;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A:** S-lukko omistajaan id = 2 (Keijo Kullervo) **Transaktio B**: Pyydetty X-lukko omistajaan id = 2 (Keijo Kullervo), jota transaktio odottaa timeoutiin asti. Tämän jälkeen lukko vapautuu.
3	**SELECT enimi, snimi FROM Omistaja WHERE id = @omistaja_id;**		
4		**COMMIT;**	
5	**SELECT enimi, snimi FROM Omistaja WHERE**		

	id = @omistaja_id;		
6	**ROLLBACK;** Query OK, 0 rows affected (0.04 sec)		

Huomioita:

Transaktio B ei onnistu poistamaan saraketta enimi vaiheessa 2 johtuen transaktion A hankkimasta S-lukosta omistajaan id = 2.

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja

id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili

id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Skenaarion kuvaus:

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET @omistaja_id = 2;

Vaihe	Transaktio A	Transaktio B
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;**	
2		**UPDATE Tili SET saldo = saldo + 11 WHERE id = 2001;**
3	**SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOM-MITTED;**	
4	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;**	
5		**COMMIT;**
6		**UPDATE Tili SET saldo = saldo + 9 WHERE id = 2001;**
7	**SET SESSION TRANSACTION ISOLATION LEVEL**	

	SERIALIZABLE;	
8	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;**	
9		**COMMIT;**
10		**UPDATE Tili SET saldo = saldo + 8 WHERE id = 2001;**
11	**SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;**	
12	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;**	
13		**COMMIT;**
14	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;**	

ISOLATION LEVEL: READ COMMITTED -> READ UNCOMMITTED -> SERIALIZABLE -> REPEATABLE READ.

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET @omistaja_id = 2;

Vaihe	**Transaktio A**	**Keijon tilien 2001, 2002 ja 2003 yhteissaldo**	**Transaktio B**
		1200	
1	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;** -> 1200	1200	
2			**UPDATE Tili SET saldo = saldo + 11 WHERE id = 2001;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0
		Committed: 1200 Uncommitted: 1211	
3	**SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;**	Committed: 1200 Uncommitted: 1211	

	Query OK, 0 rows affected (0.00 sec)		
4	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id;** -> 1211	Committed: 1200 Uncommitted: 1211	
5			**COMMIT;** Query OK, 0 rows affected (0.04 sec)
		1211	
6			**UPDATE Tili SET saldo = saldo + 9 WHERE id = 2001;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0
		Committed: 1211 Uncommitted: 1220	
7	**SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;** Query OK, 0 rows affected (0.00 sec)	Committed: 1211 Uncommitted: 1220	
8	**SELECT SUM(saldo) FROM Tili WHERE omi_id =**	Committed: 1211 Uncommitted: 1220	

	@omistaja_id ; ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction		
9			**COMMIT;** Query OK, 0 rows affected (0.06 sec)
		1220	
10			**UPDATE Tili SET saldo = saldo + 8 WHERE id = 2001;** Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0
		Committed: 1220 Uncommitted: 1228	
11	**SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;** Query OK, 0 rows affected (0.00 sec)	Committed: 1220 Uncommitted: 1228	
12	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;**	Committed: 1220 Uncommitted: 1228	

	-> 1220		
13			**COMMIT;** Query OK, 0 rows affected (0.06 sec)
14	**SELECT SUM(saldo) FROM Tili WHERE omi_id = @omistaja_id ;** -> 1220	1228	

Huomioita: Eristyvyystason voi muuttaa miksi tahansa (sekä tiukemmaksi että vähemmän tiukaksi) myös kesken transaktion. Eristyvyystasojen käyttäytyminen pysyy ennallaan.

Skenaarion kuvaus (pääskenaario 2):

use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @tili_id = 3001;

Vaihe	Transaktio A	Transaktio B
	SET SESSION TRANSACTION ISOLATION LEVEL SERIAL-IZABLE;	
		SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
	UPDATE Tili SET saldo = saldo + 100 WHERE id = @tili_id;	
		SELECT saldo FROM Tili WHERE id = @tili_id;
	COMMIT;	
		SELECT saldo FROM Tili WHERE id = @tili_id;

ISOLATION LEVEL: SERIALIZABLE (Transaktio A), READ COMMITTED (Transaktio B)

```
use Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @tili_id = 3001;
```

Vaihe	Transaktio A	Transaktio B
	SET SESSION TRANSACTION ISOLATION LEVEL SERIAL-IZABLE; Query OK, 0 rows affected (0.00 sec)	
		SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED; Query OK, 0 rows affected (0.00 sec)
	UPDATE Tili SET saldo = sal-do + 100 WHERE id = @tili_id; Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warn-ings: 0	
		SELECT saldo FROM Tili WHERE id = @tili_id; **-> 200**
	COMMIT; Query OK, 0 rows	

	affected (0.05 sec)	
		SELECT saldo **FROM Tili** **WHERE id =** **@tili_id;** **->300**

Testiä varten luotavat taulut:

```
CREATE TABLE `Tilit`.`Omistaja` (
`id` INT UNSIGNED NOT NULL ,
`enimi` VARCHAR( 20 ) NOT NULL ,
`snimi` VARCHAR( 30 ) NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

CREATE TABLE `Tilit`.`Tili` (
`id` INT UNSIGNED NOT NULL ,
`saldo` DOUBLE NOT NULL DEFAULT '0.00',
`omi_id` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = InnoDB

ALTER TABLE `Tili` ADD FOREIGN KEY ( `omi_id` ) REFERENCES `Tilit`.`Omistaja`
(
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

INSERT INTO `omistaja` VALUES (2,'Keijo','Kullervo'),(3,'Matti','Meikäläinen');

INSERT INTO `tili` VALUES (2001,200,2),(2002,300,2),(2003,700,2),(3001,100,3);
```

Taulujen alkuarvot ovat seuraavat:

Omistaja

id	enimi	snimi
2	Keijo	Kullervo
3	Matti	Meikäläinen

Tili

id	saldo	omi_id
2001	200	2
2002	300	2
2003	700	2
3001	100	3

Testatut eristyvyystasot:

		Pääskenaario 1	Pääskenaario 2
	Read Uncommitted	☑	☑
	Read Committed	☑*	☑*
	Reapeatable Read	☑*	☑*
	Serializable	☑*	☑
☑	= Testattu		
☑*	= Testattu, kuvaus vain tekstinä.		
☒	= Ei testattu/itsestäänselvä		

Skenaarion kuvaus (pääskenaario 1):

USE Tilit;
SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */
SET @tili_id = 3001;

Vaihe	Transaktio A	Transaktio B
1	**SELECT saldo FROM Tili WHERE id = @tili_id LOCK IN SHARE MODE;**	
2		**SELECT saldo FROM Tili WHERE id = @tili_id ;**
3		**UPDATE Tili SET saldo = saldo + 55 WHERE id = @tili_id;**
4		**COMMIT;**

Huomioita skenaariosta:

Transaktio A asettaa tietylle tilille S-lukon käyttämällä ekslusiivista LOCK IN SHARE MODE –lukitusta. Transaktio B:n suorittaman SELECT-lauseen tulisi onnistua ja UPDATE-lauseen epäonnistua. Tämä saattaa kuitenkin riippua käytettävästä eristyvyystasosta.

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario 1).

/*tämä testi tuottaa saman tuloksen kaikilla eristyvyystasoilla/*

USE Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SET @tili_id = 3001;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT saldo FROM Tili WHERE id = @tili_id LOCK IN SHARE MODE;** +-------+ | saldo | +-------+ | 100 | +-------+		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**SELECT saldo FROM Tili WHERE id = @tili_id ;** +-------+ | saldo | +-------+ | 100 | +-------+	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3		**UPDATE Tili SET saldo = saldo + 55 WHERE id =**	**Transaktio A**: S-lukko tiliin 3001.

		@tili_id;	**Transaktio B**: Pyydetty X-lukko tiliin 3001 odottaa transaktion A päättymistä lock timeoutiin asti, tämän jälkeen ei lukkoja.
4		**COMMIT;** Query OK, 0 rows affected (0.11 sec)	**Transaktio A:** ei näkyviä lukkoja.
5	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Tämä testi tuottaa saman tuloksen kaikilla eristyvyystasoilla. Kuten voidaan olettaa, vaiheessa 2 suoritettava SELECT-lause onnistuu mutta UPDATE-lause jää odottamaan X-lukon saamista tilille 3001.

Skenaarion kuvaus (pääskenaario 2):

USE Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET @tili_id = 3001;

Vaihe	Transaktio A	Transaktio B
1	**SELECT saldo FROM Tili WHERE id = @tili_id FOR UPDATE;**	
2		**SELECT saldo FROM Tili WHERE id = @tili_id ;**
3		**UPDATE Tili SET saldo = saldo + 55 WHERE id = @tili_id;**
4		**COMMIT;**

Huomioita skenaariosta:

Transaktio A asettaa tietylle tilille X-lukon käyttämällä ekslusiivista FOR UPDATE –lukitusta. Transaktio B:n suorittamien SELECT- ja UPDATE-lauseiden tulisi epäonnistua. Tämä saattaa kuitenkin riippua käytettävästä eristyvyystasosta.

ISOLATION LEVEL: READ UNCOMMITTED (pääskenaario 2).

/*tämä testi tuottaa saman tuloksen eristyvyystasoilla READ COMMITTED ja REPEATABLE READ/*

USE Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SET @tili_id = 3001;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT saldo FROM Tili WHERE id = @tili_id FOR UPDATE;** +-------+ | saldo | +-------+ | 100 | +-------+		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**SELECT saldo FROM Tili WHERE id = @tili_id ;** +-------+ | saldo | +-------+ | 100 | +-------+	**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
3		**UPDATE Tili SET saldo = saldo + 55 WHERE id =**	**Transaktio A**: X-lukko tiliin 3001.

		@tili_id;	**Transaktio B**: Pyydetty X-lukko tiliin 3001 odottaa transaktion A päättymistä lock timeoutiin asti, tämän jälkeen ei lukkoja.
4		**COMMIT;** Query OK, 0 rows affected (0.11 sec)	
5	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Tämä testi tuottaa saman tuloksen eristyvyystasoilla READ COMMITTED ja REPEATABLE READ. Transaktio B onnistuu vaiheessa 2 lukemaan tilin 3001 saldon siitä huolimatta, että transaktio A on aiemmin asettanut siihen kohdistuvan X-lukon. Tämä johtuu siitä, että transaktio B:n suorittama SELECT-lause ei käytä lukituksia näillä eristyvyystasoilla. Näin konfliktia lukkojen välille ei pääse syntymään. Tilanne muuttuu, mikäli transaktio B:n suorittama SELECT-lause asetetaan käyttämään lukituksia lisäämällä sen perään lause LOCK IN SHARE MODE tai FOR UPDATE. Tässä tapauksessa transaktio B yrittäisi saada tiliin 3001 S- tai X-lukkoa, jota ei voida myöntää johtuen transaktio A:n X-lukosta kyseiseen tiliin.

ISOLATION LEVEL: SERIALIZABLE (pääskenaario 2).

USE Tilit;

SET AUTOCOMMIT = 0; /* Transaktioiden yhteiset alkumäärittelyt */

SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET @tili_id = 3001;

Vaihe	**Transaktio A**	**Transaktio B**	**Lukot kyselyn suorittamisen jälkeen.**
1	**SELECT saldo FROM Tili WHERE id = @tili_id FOR UPDATE;** +-------+ | saldo | +-------+ | 100 | +-------+		**Transaktio A:** ei näkyviä lukkoja. **Transaktio B:** ei näkyviä lukkoja.
2		**SELECT saldo FROM Tili WHERE id = @tili_id ;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting trans-action	**Transaktio A**: X-lukko tiliin 3001. **Transaktio B**: Pyydetty S-lukko tiliin 3001 odot-taa transaktion A päättymistä lock timeoutiin asti, tämän jälkeen ei lukkoja.

3		**UPDATE Tili SET saldo = saldo + 55 WHERE id = @tili_id;** ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction	**Transaktio A**: X-lukko tiliin 3001. **Transaktio B**: Pyydetty X-lukko tiliin 3001 odottaa transaktion A päättymistä lock timeoutiin asti, tämän jälkeen ei lukkoja.
4		**COMMIT;** Query OK, 0 rows affected (0.11 sec)	**Transaktio A:** ei näkyviä lukkoja.
5	**ROLLBACK;** Query OK, 0 rows affected (0.00 sec)		

Huomioita:

Testin tulos eroaa alemmista eristyvyystasoista. Eristyvyystasolla SERIALIZABLE InnoDB käyttää lukituksia, joten sekä vaiheen 2 SELECT-lause että vaiheen 3 UPDATE-lause epäonnistuvat transaktio A:n vaiheessa 1 hankkimasta X-lukosta johtuen.