Transaktiot

Luodaan Kauppa -nimnen tietokanta. Tämän jälkeen luodaan siihen Tuote, Tuoteryhmä, ja Veroluokka -taulut:

```
CREATE TABLE Veroluokka (
     veroluokkaid int not null primary key identity(1,1),
     nimi varvhar(15) not null,
     veroprosentti decimal(8,2) not null
);

CREATE TABLE TuoteRyhma (
     tuoteryhmanro int not null primary key identity(1,1),
     nimi varchar(15) not null,
     veroluokkaid int null,
     FOREIGN KEY (VeroluokkaID) REFERENCES Veroluokka(veroluokkaid)
);

CREATE TABLE Tuote (
     tuotenro int not null primary key identity(1,1),
     nimi varchar(25) not null,
     hinta decimal(8,2) null,
     tuoteryhmanro int null,
     FOREIGN KEY (tuoteryhmanro) REFERENCES Tuoteryhma(tuoteryhmanro)
);
```

Seuraavaksi lisätään `veroluokka`-, `tuoteryhmä`- ja `tuote` tauluhin useita rivejä dataa kerralla useassa eri `insert into` –lauseessa. Alla nämä lauseet:

```
INSERT INTO veroluokka (veroluokkaid, nimi , veroprosentti)
          VALUES (1, 'Yleinen arvonlisävero', 22);
INSERT INTO veroluokka (veroluokkaid, nimi , veroprosentti)
          VALUES (2, 'Ruoka', 12);

INSERT INTO tuoteryhma (tuoteryhmanro, nimi , veroluokkaid)
          VALUES (10, 'Ulkoilutakit', 1);
INSERT INTO tuoteryhma (tuoteryhmanro, nimi , veroluokkaid)
          VALUES (11, Kengät, 1);


INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
          VALUES (1200, 'Protech takki', 250.5, 10);
INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
          VALUES (1300, 'All Around kengät', 150, 11);
INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
          VALUES (1450, 'Rainy sadetakki', 78.5, 10);
INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
          VALUES (1500, 'Nokia kumisaappaat', 90, 11);
```

Nyt voisimme suorittaa yllä esitetyt `insert into` lauseet siten, että talletus tietokannan tiedostoon suoritettaisiin kerralla `commit` lauseen suorituksen yhteydessä. Ei siis enää niin, että jokainen `insert into` lause suoritetaan erikseen ja tieto tallennetaan sen yhteydessä tietokannan tiedostoon. Tämän kokeilemista on tuote, tuoteryhma ja veroluokka taulut tyhjennettävä siellä olevista datariveistä `delete from` lauseella.

```
DELETE FROM TUOTE;
DELETE FROM TUOTERYHMA;
DELETE FROM VEROLUOKKA;
```

Nyt voidaan suorittaa `insert into` lauseet siten, että `commit` suorittaa lopuksi datarivien lisäämisen tietokannan tiedostoon kerralla lopuksi:

```
BEGIN TRANSACTION
INSERT INTO veroluokka (veroluokkaid, nimi , veroprosentti)
           VALUES (1, 'Yleinen arvonlisävero', 22);
INSERT INTO veroluokka (veroluokkaid, nimi , veroprosentti)
           VALUES (2, 'Ruoka', 12);

INSERT INTO tuoteryhma (tuoteryhmanro, nimi , veroluokkaid)
           VALUES (10, 'Ulkoilutakit', 1);
INSERT INTO tuoteryhma (tuoteryhmanro, nimi , veroluokkaid)
           VALUES (11, 'Kengät', 1);


INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
           VALUES (1200, 'Protech takki', 250.5, 10);
INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
           VALUES (1300, 'All Around kengät', 150, 11);
INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
           VALUES (1450, 'Rainy sadetakki', 78.5, 10);
INSERT INTO Tuote (tuotenro, nimi, hinta, tuoteryhmanro)
           VALUES (1500, 'Nokia kumisaappaat', 90, 11);
COMMIT;
```

Lopuksi suoritettiin `commit` –lause, joka suorittaa varsinaisen tallennuksen tietokannan käyttämään tiedostoon kiintolevyllä. Tietoa lisättäessä `veroluokka`-, `tuoteryhmä`- ja `tuote`- tauluihin ei ole itse asiassa paljon väliä suoritetaanko datarivien tallennus tietokannan tiedostoon rivi riviltä vai suorittaako tallenneuksen yhdellä kerralla lopuksi `commit` –lauseessa. Tilanne on kuitenkin täysin eri jos yllä esitetyssä tehtävässä olisi `tilaus`- ja `asiakas` taulut, joihin voisi tallentaa suoraan asiakkaan itse tekemiä tilauksia. Toinen esimerkki on vaikkapa pankin `tili` –taulun kohdalla. Siinä asiakkaan pitää pystyä tekemään tilisiirto tai maksamaan lasku siten, että asiakkaan tililtä vähennettään laskun edellyttämä määrä rahaa ja se listätään sen tilille, jolle laskua ollaan maksamassa. Esimerkiksi jos Pekka Peltosella on tili pankissa ja sen tilinumero 4455 ja hän haluaa maksaa 70 euron sähkölaskun Sähköisku Oy:lle. Sähköisku Oy:n tili (tilinumero on 9900) on myös samaisessa pankissa kuin Pekka Peltosenkin tili (voisi olla eri pankissakin). Tällainen siirto, jossa raha siirtyy yhdeltä asiakkaalta toiselle, on sellainen operaatio tietokannassa, että se on tehtävä kerralla alusta loppuun asti ilman virheitä. Se ei myöskään saa jäädä kesken tms. Ei tarvitse kuin kuvitella tilannetta, että laskun maksu keskeyttyy jostain syystä siten, että Pekka Peltosen tililtä ehdittiin vähentää 70 euroa mutta sitä ei ehditty lisätä Sähköisku Oy:n tilille. Tilanne olisi todella nolo pankille ja aiheuttaisi harmaita hiuksia Pekka Peltoselle ja lisäksi Sähköisku Oy luulisi ettei laskua olisi maksettu ja lähettäisi karhulaskun Pekka Peltoselle. Näin toimiva tietokanta ei missään tapauksessa olisi pankin vaatimusten mukaisesti toimiva tietokanta – eikä itse asiassa mikään muukaan yritys tai yksityinen henkilö olisi tyytyväinen tällaiseen järjestelmään. Vastaavanlaisia tilanteita löytää esimerkisi vaikkapa kaupan tilaus järjestelmistä, joiden pitää pystyä esimerkiksi vähentämään asiakkaan tilaamien tuotteiden määrä varastosta (varasto taulusta). Sama pätee lipun varausjärjestelmiin yms. Transaktioista tarvitaan siis helposti hyvinkin erilaisissa tilanteissa. Pankki esimerkki ei ole ainoa, mutta hyvin kuvaava. Pankki esimerkin tilalle voisi helposti kuvitella siis kaupan tilausjärjestelmän, lipun varausjärjestelmän, minkä tahansa järjestelmän jossa tarvitaan lukita rivejä operaation suorituksen ajaksi jotta se saadaan tehtyä asiallisesti kerralla loppun (tai peruutettua jos jokin meni pieleen).

Onneksi nykyaikaiset tietokannat pystyvät huolehtimaan, ettei yllä esitetyn kaltaista ”hävinneen” laskun tilannetta pääse milloinkaan tapahtumaan tietokannassa. Lisäksi tilauksen tai tilisiirron kaltaisten operaatioiden tarpeisiin on kehitetty ominaisuus, jolla voi suojata ettei kesken oleva tilisiirto tms. näy muille. ”Hävinneen” laskun lisäksi on nimittäin olemassa toinenkin ongelma: ”naapuri voi nähdä kesken olevan operaation tiedot”. Kuvitellaan esimerkiksi tilanne, jossa pankissa on `tili` –taulu, jossa on alla olevat tiedot:

```
Tili –taulu
tilinro    saldo
-----------------
4455       1000
9900       200000
```

Pankin `tili` –taulusta löytyy siis kaksi tiliä, yksi on Pekka Peltosen ja toinen on Sähköisku Oy:n tili. Kuvitellaan nyt tilanne, jossa tilisiirto tehdään sql:n `update` –lausetta käyttäen:

```
UPDATE Tili SET saldo = saldo - 70 WHERE tilinro = 4455;
UPDATE Tili SET saldo = saldo + 70 WHERE tilinro = 9900;
```

joka käytännössä tarkoittaa, että yllä olevat lauseet ajetaan rivi riviltä. Ensin siis vähennetään Pekka Peltosen tililtä 70 euroa ja tallennetaan tieto tietokannan tiedoston (on heti sen jälkeen muiden nähtävillä). Sen jälkeen lisätään 70 euroa Sähköisku Oy:n tilille ja tallennetaan tieto tietokannan tiedostoon (ja tämän jälkeen tämäkin tieto on kaikkien muiden nähtävillä). Lisäksi transaktiohan voidaan peruuttaa `rollback` –lauseella, päädytään vielä pahempaan tilanteeseen. Lisäksi esimerkiksi viimeinen `update` lauseen suoritus voi epäonnistua syystä tai toisesta (levy rikko, tietoliikenne ongelma, muistionglema jne). Tällöin päädytään tilanteeseen, jota ei saisi tapahtua missään tapauksessa, eli 70 euroa katoaa savuna ilmaan. Oletamme kuitenkin, että tässä ei mikään näistä pääse tapahtumaan, vaan joku pankissa ehtii ensimmäisen `update` lauseen suorituksen jälkeen laittaa alla olevan `select` lauseen suoritukseen:

```
SELECT * FROM Tili;

Tili –taulu
tilinro    saldo
-----------------
4455       930
9900       200000
```

Pankista on siis kadonnut 70 euroa. Pekka Peltosen tililtä on vähennetty 70 euroa, mutta sitä ei ole vielä ehditty lisätä Sähköisku Oy:n tilille. Seuraavaksi suoritettava `update` tekisi sen, mutta joka tapauksessa joku pääsee katsomaan kesken eräistä tilannetta. Jotain siis pitää tehdä yllä esitetyille kahdelle `update` –lauseelle. Vastaus tietokannoissa ovat transaktiot. Tietokantojen transaktioiden hallinta pitää huolen, ettei edellä esitellyn kaltaista ”’hävinnen” –laskun ongelmaa pääse tapahtumaan. Eikä myös sitä, että ”naapuri pääsee näkemään kesken eräisiä operaatiota”. Alla on esitetty miten transaktio käynnistetään ja lopetetaan:

```
BEGIN TRANSACTION
     UPDATE Tili SET saldo = saldo - 70 WHERE tilinro = 4455;
     UPDATE Tili SET saldo = saldo + 70 WHERE tilinro = 9900;
COMMIT;
```

Sql Server:ssä transaktiot aloitetaan `begin transaction` –lauseella. Tällöin yllä olevat lauseet käynnistävät transaktion Sql Server:ssä, joka päättyy `commit` –lauseeseen. `Commit` –lause siis päättää transaktion ja käytännössä avaa transaktion ajaksi lukitut datarivit. Muut eivät siis voi transaktion suorituksen aikana lukittuja rivejä lukea, muuttaa eikä poistaa. Nyt yllä transaktionhallinnan sisään laitettu laskun maksu onnistuu ilman ongelmia. Kukaan ei näe kesken eräistä siirtoa, joka voidaan peräti keskeyttää `rollback` -lauseella (josta muutoin seuraisi ns. dirty read problem). Vrhetilanteetkaan eivät enää transaktionhallinnan ansiosta aiheuta tiedon vääristymistä `tili` taulussa esimerkiksi tilisiirron kaltaisissa tietokantaoperaatioissa.

Onkin suositeltavaa suorittaa sql lauseet aina yllä olevan esimerkin mukaisesti transaktionhallinnan alaisuudessa. On tehokkaampaan suoritaa muutokset kerralla tietokantaan kuin lause ja rivi riviltä. Lisäksi useissa tilanteissa halutaan välttyä kesken eräisen operaation ongelmalta. Myös virhetilanteet tietokannassa voivat aiheuttaa ongelmia jos osa lauseista saadaan ajettua ennen virhetilannetta, mutta loput lauseista jäävät suorittamatta.

Transaktioidenhallinnan käytölle siis löytyy tärkeitä perusteita, eikä tässä ole edes kaikki. Näistä asioista tulee lisää myöhemmin järjestettävissä tietokantojen opintojaksoilla.