CS-A1150 Tietokannat

2.4.2019

Oppimistavoitteet: tämän luennon jälkeen

- Tiedät, miten tietokannan taulujen (relaatioiden) määrittelyt kirjoitetaan SQL:llä.
- Osaat päivittää tietokannan tauluja ja niiden sisältöjä SQL:llä.
- Osaat määritellä SQL:ssä eheysehtoja.
  - Miten voidaan esimerkiksi määritellä, että jonkin attribuutin arvon pitää esiintyä toisessa relaatiossa jonkin monikon avainattribuutin arvona?
  - Mitä tietokannan hallintajärjestelmä tekee, jos vaadittu eheysehto ei toteudu?
- Tiedät, mitä ovat näkymät (views) ja kuinka niitä määritellään ja käytetään.

Esimerkkitietokanta

- Tämän luennon esimerkit käsittelevät aikaisempien luentojen esimerkkitietokantaa, joka koostuu seuraavista relaatioista

  `Customers(custNo, name, born, bonus, address, email)`
  `Products(number, prodName, description, price, manufID)`
  `Manufacturers(ID, manufName, phone)`
  `Orders(orderNo, deliver, status, custNo)`
  `BelongsTo(orderNo, productNo, count)`

Relaatioiden määrittely SQL:llä

- SQL:ssä voidaan esittää kolmenlaisia relaatioita
  1. *Taulut* (tables). Nämä ovat pysyvään muistiin tallennettuja relaatioita.
  2. *Näkymät* (views). Relaatioita, joita ei tallenneta pysyvään muistiin, mutta joita voidaan määritellä ja jotka luodaan tarvittaessa (esim. kun niihin kohdistetaan kysely).
  3. *Väliaikaiset taulut* (temporary tables), joita ei tallenneta lainkaan, mutta joita kyselynkäsittelijä luo tarvittaessa esimerkiksi kyselyn välivaiheita varten. Nämä tuhotaan heti, kun niitä ei tarvita.

Taulun määrittely SQL:ssä

- Taulu määritellään CREATE TABLE -käskyllä, esimerkiksi *Customers*-relaatiota vastaava taulu voitaisiin luoda käskyllä

```
CREATE TABLE Customers (
    custNo CHAR(10),
    name CHAR(100),
    born INT,
    bonus INT,
    address CHAR(100),
    email CHAR(100)
);
```

- Käskyssä on siis kerrottu relaation nimi sekä attribuuttien nimet ja tyypit. Kaikilla attribuuteilla täytyy olla jokin tyyppi.

Mahdollisia attribuuttien tyyppejä

- Merkkijonoille CHAR(n) *n*:n merkin mittaisille merkkijonoille (merkkijono voi olla myös tätä lyhyempi, mutta sille varataan taulussa aina *n*:n merkin vaatima tila), ja VARCHAR(n) vaihtelevan mittaisille, mutta korkeintaan *n*:n merkin mittaisille merkkijonoille (tietokannan hallintajärjestelmä voi varata tällöin merkkijonolle vain niin ison tilan kuin mitä merkkijono oikeasti tarvitsee). Merkkijonovakiot esitetään yksinkertaisissa lainausmerkeissä, esimerkiksi 'foo'.
- Bittijonoille BIT(n) ja BIT VARYING(n) (ensimmäisessä tapauksessa jonolle varataan aina *n*:n bitin kokoinen tila, toisessa voidaan varata bittijonon todellisen pituuden mittainen, mutta korkeintaan *n*:n bitin pituinen tila.)
- Totuusarvoille BOOLEAN. Tyypin mahdolliset arvot ovat TRUE, FALSE ja UNKNOWN.

Mahdollisia attribuuttien tyyppejä, jatkoa

- Kokonaisluvuille INT ja INTEGER. Nämä tarkoittavat samaa. Lisäksi tyyppi SHORTINT, jolle saattaa olla varattu pienempi tila (riippuu toteutuksesta).
- Liukuluvuille FLOAT ja REAL (synonyymeja). Lisäksi DOUBLE PRECISION, joka voi olla tarkempi, sekä DECIMAL(n,d), joka määrittelee luvussa esiintyvien numeroiden ja desimaalien määrän, esimerkiksi 0123.45 olisi tyypin DECIMAL(6,2) mukainen (tallennettu tarkkana arvona). NUMERIC on suunnilleen edellisen synonyymi, mutta joissakin ympäristössä toteutuksessa voi olla joitakin eroja.
- Päivämääriä varten tyyppi DATE ja kellonaikoja varten tyyppi TIME. Molempia esitetään merkkijonoina. Eri ympäristöt voivat tarjota erilaisia tapoja saman päivämäärän tai kellonajan kuvaamiseen, mutta SQL-standardissa päivämääriä kuvataan tyylillä DATE '1948−05−14' (14.5.1948) ja kellonaikoja TIME '15:00:02.5' (kaksi ja puoli sekuntia yli 15). Päivämääriä voi vertailla tavallisten vertailuoperaattoreiden avulla.

Esimerkki

- Määritellään relaatiota

  `Products(number, prodName, description, price, manufID)`

  vastaava taulu SQL:llä.
- Mahdollinen määrittely:

```
CREATE TABLE Products(
    number CHAR(10),
    prodName CHAR(80),
    description VARCHAR(200),
    price REAL,
    manufID CHAR(10)
);
```

Huomatus tyypeistä

- Todellisuudessa käytössä olevat tyypit vaihtelevat käytettävän tietokannan hallintajärjestelmän mukaan.
- Esimerkiksi kurssilla käytettävä SQLitessa on vain tyypit INTEGER, REAL, TEXT ja BLOB. SQLite ei tunne lainkaan tyyppiä BOOLEAN, vaan totuusarvoja esitetään lukujen 1 (true) ja 0 (false) avulla.
- SQLite käyttää dynaamista tyypitystä eikä tarkista, että attribuuteille annetaan määrittelyjen mukaisia arvoja.
- SQLite kuitenkin hyväksyy standardin mukaiset määrittelyt, mutta muuttaa niissä annetut tyypit joksikin sen käyttämistä.
- Lisätietoa SQLiten tyypeistä sivulla `http://www.sqlite.org/datatype3.html`.

Päivämäärät SQLitessä

- SQLitessä päivämääriä voidaan ilmaista merkkijonoina, esimerkiksi '2008−05−14' tarkoittaa 14.5.2008 ja '2008−05−14 10:52' tarkoittaa 14.5.2008 klo 10:52.
- Katso muita mahdollisuuksia sivulta `http://www.sqlite.org/lang_datefunc.html`
- Päivämääriä voi verrata vertailuoperaattoreiden avulla, esim.

```
SELECT *
FROM MovieStar
WHERE birthdate >= '1970−01−01';
```

  hakee harjoituskierroksen 2 `MovieStar`-relaatiosta vuonna 1970 tai sen jälkeen syntyneet näyttelijät.
- **Huom:** Päivämäärissä erotinmerkkinä esim. vuoden ja kuukauden välissä on tavallinen tavuviiva. Jos kopioit päivämääriä leikkaamalla ja liimaamalla suoraan tästä pdf-tiedostosta, voi erotinmerkiksi tulla väärä merkki.

Tietokantakaavioiden muuttaminen SQL:ssä

- Mahdollisia toimenpiteitä
  - Uuden taulun lisääminen tietokantaan.
  - Taulun poistaminen tietokannasta.
  - Uuden attribuutin lisääminen tauluun.
  - Attribuutin poistaminen taulusta.

Tietokantakaavioiden muuttaminen SQL:ssä, jatkoa

- Tauluja voidaan poistaa käskyllä DROP TABLE. Esimerkiksi

  DROP TABLE Customers;

- Olemassa olevaan tauluun voidaan lisätä ja siitä voidaan poistaa attribuutteja (sarakkeita) käskyllä ALTER TABLE, esim.

  ALTER TABLE Customers ADD phone CHAR(16);

  lisää `Customers`-tauluun attribuutin `phone` (tyyppinä 16 merkin pituinen merkkijono), ja

  ALTER TABLE Customers DROP born;

  poistaa `Customers`-taulusta attribuutin `born` (jälkimmäinen ei toimi SQLitessä).

Oletusarvojen määrittely

- Uusia tauluja määriteltäessä ja lisättäessä attribuutteja vanhoihin tauluihin voidaan määritellä attribuuteille oletusarvoja. Jos taulun jollakin monikolla attribuutille ei ole annettu muuta arvoa, niin käytetään oletusarvoa.
- Oletusarvo voi olla joko `NULL` tai jokin vakio.

Oletusarvojen määrittely, esimerkkejä

- Esimerkki taulua määriteltäessä annetusta oletusarvosta (oletusarvot annettu bonukselle ja osoitteelle):

```
CREATE TABLE Customers (
    custNo CHAR(10),
    name CHAR(100),
    born INT,
    bonus INT DEFAULT 0,
    address CHAR(100) DEFAULT '?',
    email CHAR(100)
);
```

- Esimerkki oletusarvon antamisesta tauluun lisättävälle attribuutille:

```
ALTER TABLE Customers ADD phone CHAR(16) DEFAULT 'unknown';
```

Avainten määrittely

- SQL:ssä on kaksi eri tapaa määritellä, että jokin attribuutti tai joukko attribuutteja on taulun avain:
  1. Attribuutti määritellään avaimeksi siinä kohdassa, jossa attribuutti ja sen tyyppi listataan taulun määrittelyssä.
  2. Taulun määrittelyssä attribuuttien listan jälkeen kerrotaan, mitkä attribuutit muodostavat taulun avaimen.

  Jos avain koostuu yhdestä attribuutista, voidaan käyttää kumpaa tapaa tahansa. Jos taas avain koostuu useammasta attribuutista, on käytettävä jälkimmäistä tapaa.
- Jos on määritelty taulun R avain, niin tietokannan hallintajärjestelmä ei anna lisätä tauluun kahta monikkoa, joilla olisi sama avainarvo.
- Avaimen muodostavat attribuutit voidaan määritellä joko määreen PRIMARY KEY tai määreen UNIQUE avulla.

Avainten määrittely, jatkoa

- Määreellä PRIMARY KEY määritellyille attribuuteille ei voi antaa arvoksi `NULL`-arvoja, määreellä UNIQUE määritellyille attribuuteille voi.
  - Poikkeus SQLitessä: SQLitessä myös PRIMARY KEY määritellyille attribuuteille voi antaa arvoksi `NULL`-arvoja, jos niitä ei ole muilla eheysehdoilla kielletty.
- Jos attribuutti on määritelty UNIQUE-määreellä, usealla saman relaation attribuutilla voi kuitenkin olla attribuutin arvona `NULL`, vaikka muuten samat arvot eivät ole sallittuja.

Avainten määrittely, esimerkki

- Ensimmäinen tapa:

```
CREATE TABLE Customers (
    custNo CHAR(10) PRIMARY KEY,
    name CHAR(100),
    born INT,
    bonus INT,
    address CHAR(100),
    email CHAR(100)
);
```

- Toinen tapa:

```
CREATE TABLE Customers (
    custNo CHAR(10),
    name CHAR(100),
    born INT,
    bonus INT,
    address CHAR(100),
    email CHAR(100),
    PRIMARY KEY (custNo)
);
```

Avainten määrittely, toinen esimerkki

- Jos halutaan määritellä taulu `BelongsTo`, jossa tilausnumero ja tuotenumero muodostavat yhdessä avaimen, on avaimen määrittelyssä käytettävä jälkimmäistä tapaa:

```
CREATE TABLE BelongsTo(
    orderNo CHAR(10),
    productNo CHAR(10),
    count INTEGER,
    PRIMARY KEY (orderNo, productNo)
);
```

Monikoiden lisääminen tauluun

- Tauluun voidaan lisätä uusia monikoita (rivejä) käskyllä INSERT INTO.
- Esimerkki:

  INSERT INTO Products(number, prodName, description, price, manufID)
  VALUES('R−55336', 'IPad Air 2', 'tablet', 495.0, 'M554');

  Tässä on siis taulun nimen jälkeen lueteltu taulun attribuutit ja sen jälkeen attribuuteille annettavat arvot samassa järjestyksessä. Kaikille relaation attribuuteille ei ole pakko antaa arvoa (nämä attribuutit jätetään pois listasta).
- Käskyssä ei tarvitse luetella attribuuttien nimiä, jos kaikille taulun attribuutteille annetaan arvot ja arvot luetellaan samassa järjestyksessä kuin attribuuttien nimet on annettu taulua luodessa:

  INSERT INTO Products
  VALUES('R−55336', 'IPad Air 2', 'tablet', 495.0, 'M554');

Monikoiden lisääminen tauluun, jatkoa

- Tauluun voidaan lisätä kaikki jonkin kyselyn tuloksena tulevat monikot.
- Oletetaan, että on määritelty erilaisille tuoteryhmille taulu `Productgroups` relaatiolle

  `Productgroups(name, supergroup)`

  ja että taulu on tällä hetkellä tyhjä. Halutaan lisätä tauluun ryhmien nimiksi kaikki kuvaukset, jotka esiintyvät `Products`-taulussa:

```
INSERT INTO Productgroups(name)
    SELECT DISTINCT description
    FROM Products;
```

Monikoiden poistaminen taulusta

- Taulusta voidaan poistaa monikoita käskyllä

  DELETE FROM taulu
  WHERE ehto;

  Käsky poistaa taulusta `taulu` kaikki ne monikot, jotka toteuttavat ehdon `ehto`.

- Esimerkki: poistetaan `Products`-taulusta kaikki ne tuotteet, joiden valmistajan tunnus on F542:

  DELETE FROM Products
  WHERE manufID = 'F542';

Monikoiden päivitykset

- Käskyllä UPDATE voidaan päivittää taulussa olevan monikon attribuutin arvoa, että monikkoa tarvitsee poistaa taulusta tai tauluun tarvitsee lisätä uusia monikoita.
- Käskyn muoto on

  UPDATE taulu
  SET uusien arvojen sijoitukset
  WHERE ehto;

  Uudet arvot annetaan sijoituskäskyjen avulla (attribuutti = arvo).
- Esimerkki: vaihdetaan tuotteen kuvaukseksi computer kaikilla sellaisilla tuotteilla, jolla se tällä hetkellä on pc:

  UPDATE Products
  SET description = 'computer'
  WHERE description = 'pc';

Eheysehdot ja laukaisimet

- Ohjelmoija voi määritellä SQL-tietokantaan erilaisia *eheysehtoja* (integrity constraints), jotka määrittelevät tietokannan sallittuja tiloja, esimerkiksi
  - Määrätty attribuuttijoukko on relaation avain (relaation kahdella monikolla ei saa olla samaa arvoa tälle attribuuttijoukolle)
  - Yhden relaation jonkin attribuutin arvon pitää esiintyä toisen relaation jonkin monikon avainarvona (viite-eheys)
  - Attribuutin arvon on oltava määrätyllä välillä
  - Monikon kahden tai useamman attribuutin arvon pitää toteuttaa toisiinsa nähden annettu ehto.
  - Useamman monikon tai relaation välillä täytyy olla voimassa jokin ehto.
- Lisäksi tietokantaan voi määritellä *laukaisimia* (triggers). Ne ovat toimenpiteitä, jotka tietokannan hallintajärjestelmä käynnistää automaattisesti, kun määrätty tapahtuma sattuu. Näitä käsitellään myöhemmällä luennolla.

Eheysehdot, jatkoa

- SQL-standardissa on määritelty, millaisia eheysehtoja voidaan kirjoittaa. Eri toimittajien tietokannan hallintajärjestelmät vaihtelevat kuitenkin suuresti siinä, mitä standardissa määriteltyjä ominaisuuksia ne toteuttavat.
- Miksi käyttää eheysehtoja?
  - Havaitaan, jos tietokantaan yritetään syöttää vääränlaista dataa (esim. kirjoitusvirheet datan syöttämisen tai päivitysten yhteydessä)
  - Pidetetään huolta siitä, että data säilyy konsistenttina (kun esim. yhden relaation monikosta viitataan toisen relaation monikkoon).
  - Kerrotaan tietokannan hallintajärjestelmälle tallennettavan datan luonteesta. (Esim. tieto siitä, että attribuutti on relaation avain, voi vaikuttaa siihen, miten tietokannanhallintajärjestelmä tallentaa relaation monikot.)

Viite-eheys ja viiteavaimet

- Määrätään, että yhden relaation attribuutti tai attribuuttijoukko esiintyy toisen relaation avainarvona.
- Esimerkki: määrätään, että jokainen *Products*-relaation `manufID`-attribuutin arvo esiintyy *Manufacturers*-relaation jonkin monikon `ID`-attribuutin arvona.
- Tällöin sanotaan, että `manufID`-attribuutti on *viiteavain* (foreign key).

Viiteavaimen määrittely

- Viiteavain voidaan määritellä joko CREATE TABLE -komennossa samalla, kun taulun attribuutteja luetellaan:

```
CREATE TABLE Products(
        number CHAR(10) PRIMARY KEY,
        prodName CHAR(80),
        description VARCHAR(200),
        price REAL,
        manufID CHAR(10) REFERENCES Manufacturers(ID)
);
```

Viiteavaimen määrittely, jatkoa

- Toinen vaihtoehto on määritellä viiteavain erikseen FOREIGN KEY-määreellä attribuuttien luettelemisen jälkeen:

```
CREATE TABLE Products(
    number CHAR(10) PRIMARY KEY,
    prodName CHAR(80),
    description VARCHAR(200),
    price REAL,
    manufID CHAR(10),
    FOREIGN KEY (manufID) REFERENCES Manufacturers(ID)
);
```

- **Jos viiteavain muodostuu useasta attribuutista, on käytettävä jälkimmäistä tapaa. Muuten määrittely ei vaadi, että viiteavaimen kaikkien attribuuttien arvojen pitää esiintyä yhdessä avaimena samassa monikossa.**

Välitehtävä

Tarkastellaan taulua `StarsIn` (harjoituskierros 2).
Mitä eroa on seuraavilla määrittelyillä:

```
CREATE TABLE StarsIn(
    movieTitle CHAR(100) REFERENCES Movies(title),
    movieYear INT REFERENCES Movies(year),
    starName CHAR(30) REFERENCES MovieStar(name)
);
```

```
CREATE TABLE StarsIn(
    movieTitle CHAR(100),
    movieYear INT,
    starName CHAR(30),
    FOREIGN KEY (movieTitle, movieYear) REFERENCES Movies(title, year),
    FOREIGN KEY (starName) REFERENCES MovieStar(name)
);
```

Kumpi on oikein vai ovatko molemmat?

Välitehtävän ratkaisu

- Vain jälkimmäinen vaihtoehto on oikein. Ensimmäinen vaatii, että elokuvan nimen pitää esiintyä jossain `Movies`-relaation monikossa ja elokuvan vuoden jossain `Movies`-relaation monikossa, mutta hyväksyy sen, että nimi ja vuosi esiintyvät eri monikoissa (jolloin kysymys ei ole samasta elokuvasta). Jälkimmäinen vaatii, että nimen ja vuoden pitää esiintyä samassa `Movies`-relaation monikossa.

Viite-eheyden ylläpito

- Olkoon relaatiot R ja S, jossa R:n attribuutti A on viiteavain, joka viittaa S:n attribuuttiin B. Mitkä tilanteet voivat aiheuttaa viite-eheyden rikkoutumisen?
- Vaaratilanteita
  - R:ään yritetään lisätä uusi monikko, jonka A-attribuutin arvo ei esiinny missään S:n monikossa B-attribuutin arvona.
  - R:n jonkin monikon A-attribuutin arvoa yritetään päivittää arvoksi, joka ei esiinny missään S:n monikossa B-attribuutin arvona.
  - S:stä yritetään poistaa monikko, jonka B-attribuutin arvo esiintyy jonkin R:n monikon A-attribuutin arvona.
  - S:ssä yritetään päivittää jonkin monikon B-attribuutin arvoa, vaikka arvo esiintyy jonkin R:n monikon A-attribuutin arvona.

Viite-eheyden ylläpito

- Mitä tietokannan hallintajärjestelmä tekee, jos viite-eheys uhkaa rikkoutua jonkin edellisellä kalvolla mainitun toimenpiteen takia?
- Kahdessa ensimmäisessä tapauksessa ainoa vaihtoehto on estää R:ään (relaatio, jossa viiteavain on) tuleva lisäys tai päivitys. Sen sijaan kahdessa jälkimmäisessä tapauksessa (muutetaan relaatiota, johon viiteavain viittaa) on kolme eri vaihtoehtoa:
  1. *Estopolitiikka:* Estetään S:n monikon poisto tai päivitys. Tämä on SQL:ssä oletuksena.
  2. *Ketjupolitiikka* (the cascade policy): Jos S:ssä suoritetaan muutoksia, niin myös R:ssä suoritetaan tarpeelliset muutokset, joiden jälkeen viite-eheys on jälleen voimassa. Jos esimerkiksi S:stä poistetaan monikko, niin R:stä poistetaan kaikki siihen viittaavat monikot. Jos taas jonkin S:n monikon B-attribuutin arvoa muutetaan, niin kaikissa siihen viittaviissa R:n monikon A-attribuutin arvoa muutetaan vastaavasti.
  3. *Nolla-arvopolitiikka* (the set-null policy): Jos viitatusta relaatiosta poistetaan tai muutetaan arvoa (avainta), johon on viittaus, niin viittaavassa relaatiossa viiteavaimen arvoksi asetetaan NULL.

Viite-eheyden ylläpito, jatkoa

- Käytettävä politiikka voidaan määritellä erilaiseksi poistojen ja päivitysten osalta. Politiikka määritellään samalla, kun jokin attribuutti määritellään viiteavaimeksi, esimerkiksi:

```
CREATE TABLE Products(
    number CHAR(10) PRIMARY KEY,
    prodName CHAR(80),
    description VARCHAR(200),
    price REAL,
    manufID CHAR(10) REFERENCES Manufacturers(ID)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

Estopolitiikkaa ei tarvitse määritellä, koska se on oletuksena.

Attribuuttien arvojen rajoittaminen

- SQL antaa mahdollisuuden rajoittaa attribuuttien arvoja. Yksinkertaisin rajoitus on vaatia, että attribuutin arvo ei saa olla NULL. Tämä voidaan tehdä taulua luodessa attribuutteja luetellessa (osa riveistä jätetty näyttämättä):

```
CREATE TABLE Products (
    ...
    price REAL NOT NULL,
    manufID CHAR(10) REFERENCES Manufacturers(ID) NOT NULL
);
```

- Tällöin tietokannan hallintajärjestelmä ei salli lisäyksiä tai päivityksiä, joissa näin määriteltyjen attribuuttien (esimerkissä `price` ja `manufID`) arvoksi asetettaisiin NULL. Esimerkkitapauksessa ei myöskään voi käyttää nolla-arvopolitiikkaa viiteavaimen kohdalla.

Attribuuttien arvojen rajoittaminen, jatkoa

- Arvoja voidaan rajoittaa myös muilla ehdoilla käyttämällä avainsanaa CHECK.
- Kaksi esimerkkiä (osa riveistä jätetty pois, toisessa uusi attribuutti aikaisempaan esitettynä):

```
CREATE TABLE Products (
    ...
    price REAL CHECK (price > 0.0 AND price < 5000.0),
    manufID CHAR(10) REFERENCES Manufacturers(ID) NOT NULL
);

CREATE TABLE Customers (
    ...
    gender CHAR(1) CHECK (gender IN ('F', 'M', 'O'))
);
```

Attribuuttien arvojen rajoittaminen, jatkoa

- Vaihtoehtoisesti on mahdollista määritellä arvojoukko erikseen ja käyttää taulun määrittelyssä tätä arvojoukkoa (ei toimi SQLitessä):

```
CREATE DOMAIN GenderDomain CHAR(1)
    CHECK (VALUE IN ('F', 'M', 'O'));
```

jolloin taulun määrittelyssä attribuutteja luetellessa voidaan kirjoittaa

```
CREATE TABLE Customers (
    ...
    gender GenderDomain
);
```

Globaalit rajoitteet

- Globaaleilla rajoitteilla voidaan määritellä
  - Monikkokohtaisia rajoitteita (tuple-based check constraints), joilla asetetaan ehtoja saman monikon eri attribuuteille.
  - Relaation sisäisiä ja relaatioiden välisiä yleisiä ehtoja (assertions, general constraints).
- Esimerkki: asetaan *Customers*-relaatiolle ehto, jonka mukaan asiakkaan nimi ei voi alkaa 'Ms.':llä, jos asiakas ei ole nainen:

```
CREATE TABLE Customers (
    custNo CHAR(10) PRIMARY KEY,
    gender CHAR(1),
    ....
    CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
);
```

Globaalit rajoitteet, jatkoa

- Taulua R luodessa annetussa monikkokohtaisessa rajoitteessa esiintyvä ehto voi sisältää mitä tahansa, mitä voi esiintyä tavallisen kyselyn WHERE-osassa, esimerkiksi alikyselyjä.
- Jos rajoitteen alikyselyssä esiintyy toinen relaatio S, tarkistetaan ehdon voimassaolo vain silloin, kun tauluun R tehdään muutoksia, ei silloin, kun tauluun S tehdään muutoksia.
- Esimerkki (ei toimi SQLitessä): asetetaan relaatioon *Products* ehto, jonka mukaan minkään valmistajien tuotteiden yhteishinta ei saa ylitää 10000 euroa (ei toimi SQLitessä):

```
CREATE ASSERTION SumPrice CHECK (10000.0 >= ALL
    (SELECT SUM(price) FROM Products GROUP BY manufID)
);
```

- Koska yllä annettu ehto koskee vain yhtä relaatiota, sen olisi voinut myöskin esittää monikkokohtaisena rajoitteena taulua luodessa.

Huomautus rajoitteista

- Tällä hetkellä suurin osa tietokannan hallintajärjestelmistä (esim. SQLite) ei tue kyselyitä CHECK-osassa, vaikka ne SQL-standardin mukaan ovat mahdollisia.

Rajoitteiden muuttaminen

- Rajoitteen muuttaminen tai poistaminen on mahdollista vain, jos rajoitteelle on sitä luodessa annettu nimi.
- Rajoitteelle voidaan antaa nimi lisäämällä rajoitetta määritellessä avainsana CONSTRAINT ja sen jälkeen ehdon nimi. Esimerkkejä:

```
CREATE TABLE Products (
    ...
    price REAL CONSTRAINT CorrectPrice
                CHECK (price > 0.0 AND price < 5000.0)
);

CREATE TABLE Customers (
    custNo CHAR(10) CONSTRAINT NumberKey PRIMARY KEY,
    gender CHAR(1),
    ....
    CONSTRAINT RightTitle
        CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
);
```

Rajoittteiden muuttaminen, jatkoa

- Rajoitteita muutetaan käyttämällä avainsanoja ALTER TABLE ja joko avainsanaa DROP tai ADD (eivät toimi SQLitessä).
- Esimerkkejä:

```
ALTER TABLE Customers DROP CONSTRAINT NumberKey;

ALTER TABLE Customers ADD CONSTRAINT NumberKey
    PRIMARY KEY (custNo);

ALTER TABLE Customers DROP CONSTRAINT RightTitle;
```

- Käskyllä CREATE ASSERTION luotuja rajoitteita voi poistaa DROP ASSERTION-käskyllä, esimerkiksi

```
DROP ASSERTION SumPrice;
```

Näkymät

- Taulujen lisäksi tietokantaan voidaan määritellä näkymiä (views). Niitä ei tallenneta pysyvään muistiin, vaan niiden sisältö luodaan tarvittaessa (esim. kun niihin kohdistetaan kysely).
- Näkymä määritellään komennolla

  CREATE VIEW nimi AS määritelmä;

  Missä nimi on määriteltävän näkymän nimi ja määritelmä on jokin SQL-kysely.
- Esimerkki: luodaan näkymä, joka sisältää *Products*-relaatiosta vain ne tuotteet, joiden valmistajan nimi on Samsung:

```
CREATE VIEW SamsungProducts AS
    SELECT number, prodName, description, price, manufID
    FROM Products, Manufacturers
    WHERE manufName = 'Samsung' AND manufID = ID;
```

Näkymät, jatkoa

- Toinen esimerkki: luodaan näkymä, jossa tuotteen numero ja nimi on esitetty yhdessä tuotteen valmistajan nimen kanssa:

```
CREATE VIEW ProdManuf AS
    SELECT number, prodName, manufName
    FROM Products, Manufacturers
    WHERE manufID = ID;
```

- Näkymiin voidaan kohdistaa kyselyitä ihan samalla tavalla kuin tauluihin. Samassa kyselyssä voidaan käsitellä sekä tauluja että näkymiä.
- Esimerkki: Haetaan niiden tilausten numerot, joissa esiintyy Samsungin tuotteita.

```
SELECT DISTINCT orderNo
FROM SamsungProducts, BelongsTo
WHERE number = productNo;
```

Attribuuttien uudelleennimeäminen

- Näkymää määritellessä on mahdollista antaa sen attribuuteille uudet nimet suluissa näkymän nimen jälkeen.
- Esimerkki:

```
CREATE VIEW ProdManuf(productNumber, productName, ManufacturerName) AS
    SELECT number, prodName, manufName
    FROM Products, Manufacturers
    WHERE manufID = ID;
```

- Näkymä on sama kuin aikaisemmassa esimerkissä, mutta sen sarakkeilla on nyt eri nimet. (Ei testatessa toiminut SQLitessä, vaikka dokumentaation mukaan sen pitäisi.)

Näkymän muuttamisesta

- Aikaisemmin määritelty näkymä voidaan poistaa käskyllä DROP VIEW, esimerkiksi

  DROP VIEW ProdManuf;

  Tämä poistaa vain näkymän määrittelyn, ei alkuperäisiä tauluja, joiden perusteella näkymä on luotu.
- Jos näkymä on tarpeeksi yksinkertainen (sääntöjä ei käydä läpi tällä kurssilla), on mahdollista kirjoittaa SQL-käskyjä, jotka päivittävät näkymää (lisäävät, poistavat tai päivittävät näkymään kuuluvia monikoita). Tällöin päivitykset tehdään oikeasti siihen tauluun, jonka perusteella näkymä on tehty.
- Näkymiä voidaan myös *materialisoida*, millä tarkoitetaan näkymän tallentamista ja sitä, että sitä päivitetään aina tarvittaessa. Tällä kurssilla ei käydä näkymiä läpi sen tarkemmin, mutta seuraavalla luennolla käsitellään hakemistoja, joita voidaan myös pitää materialisoituina näkyminä.