SQL - Tietokannan ylläpito

- SQL sisältää operaatiot tietokannan sisällön muodostamiseen ja ylläpitoon:
- insert - uusien rivien vienti tauluun
- delete - rivien poisto
- update - rivien muutos

SQL - Tietokannan ylläpito

- Insert lauseella on kaksi muotoa:
- insert into *taulu* [(*sarakenimet*)] values (*arvot*)
  - tällä muodolla lisätään yksi rivi ja arvot annetaan vakioina tai vakioihin perustuvina lausekkeina
- insert into *taulu* [(*sarakenimet*)] *kysely*
  - *tällä muodolla kyselyn tulosrivit lisätään tauluun*

SQL - Tietokannan ylläpito

```
CREATE TABLE kurssi (
        koodi  numeric(8) NOT NULL ,
        nimi  varchar(40) NOT NULL ,
        opintoviikot  numeric(5,1) NOT NULL ,
        luennoija  varchar(12),
        PRIMARY KEY (koodi ),
        FOREIGN KEY (luennoija) REFERENCES
  opettaja)

insert into kurssi values
  (1234,'Tietokantojen perusteet',2,'HLAINE');
```

- lisää tauluun kokonaisen rivin

SQL - Tietokannan ylläpito

- Jos luennoijaa ei tiedetä voidaan lisäys tehdä seuraavasti:

```
insert into kurssi values
  (1234,'Tietokantojen perusteet',2,NULL); tai

insert into kurssi (koodi, nimi, opintoviikot)
  values (1234,'Tietokantojen perusteet',2);
```

- **sarakeluetteloa käytetään siis silloin kun annetaan vain osa sarakkeista**

SQL - Tietokannan ylläpito

- Jos luennoijan tunnusta ei tiedetä, voitaisiin lisäys tehdä seuraavasti (kaikki kyselyn tulosrivit lisätään):

```
insert into kurssi
  select (1234, 'Tietokantojen perusteet', 2,opetunnus)
  from opettaja
  where nimi='Laine Harri' ;
```

- Tämä toimii odotetusti, jos kannassa on vain yksi tämän niminen opettaja, muuten lisäys kaatuu avaimen yksikäsitteisyysvirheeseen, sillä lisäys epäonnistuu, jos se rikkoo eheysehtoja

SQL - Tietokannan ylläpito

- Kurssille 'Ohjelmoinnin perusteet' ilmoittautuneiden opiskelijoiden siirto kurssin 'Java-ohjelmointi' vastaaviin ryhmiin:

```
CREATE TABLE ilmoittautuminen (
      kurssikoodi numeric(8) not null,
      ryhmanro numeric(2) not null,
      opisknro  numeric(5) NOT NULL ,
      ilm_aika  date NOT NULL ,
      PRIMARY KEY (opisknro, kurssikoodi) ,
      FOREIGN KEY (kurssikoodi, ryhmanro)  REFERENCES
          harjoitusryhma on delete cascade,
      FOREIGN KEY (opisknro) REFERENCES opiskelija )
```

SQL - Tietokannan ylläpito

Oracle-SQL:llä

```
Insert into ilmoittautuminen
  select java.kurssikoodi, ryhmanro, opisknro, sysdate
  from kurssi java, kurssi ohpe, ilmoittautuminen
  where java.nimi='Java-ohjelmointi' and
        ohpe.nimi='Ohjelmoinnin perusteet' and
        ohpe.koodi=ilmoittautuminen.kurssikoodi;
```

SQL - Tietokannan ylläpito

- Rivien muutokset (update)

```
update taulu
set sarake1=lauseke1 [, …]
[where kohteen rajausehdot]
```

– samalla kertaa voi muttaa useita sarakkeiden sisältöä,
– muutetaan kaikki where-ehdon täyttävät rivit
– jos ehto puuttuu muutetaan kaikki taulun rivit

SQL - Tietokannan ylläpito

- Muutetaan kurssin Java-ohjelmointi opintoviikkomäärä kolmeksi

```
update kurssi
set opintoviikot=3
where nimi='Java ohjelmointi';
```

– Muutos epäonnistuu, jos se rikkoo eheysehtoja.

SQL - Tietokannan ylläpito

- Rivien poisto (delete)

```
delete from taulu
[where poistettavien rajausehdot ];
```

– Poistetaan kaikki ehdon täyttävät rivit
– Jos ehto puuttuu poistetaan kaikki rivit
– Poisto epäonnistuu jos eheysehdot rikkoutuvat (ellei muuta ole määritelty)

SQL - Tietokannan ylläpito

- Poistetaan harjoitusryhmät joihin ei ole ilmoittautuneita:

```
delete from harjoitusryhma
where (kurssikoodi, ryhmanro) not in
     (select kurssikoodi, ryhmanro
      from ilmoittautuminen);
```

SQL - Tietokannan ylläpito

- Rivien siirtoa taulusta toiseen tarvitaan esimerkiksi siirrettäessä tietoja aktiivisesta taulusta historiatauluihin. Tämä suoritetaan kopioimalla (lisäämällä) rivit kohdetauluun ja sen jälkeen poistamalla ne lähtötaulusta:

```
insert into ilmohistoria
select * from ilmoittautumiset where ilm_aika<'1.1.1999';
delete from ilmoittautumiset where ilm_aika<'1.1.1999';
```

SQL - Tietokantatapahtuma (transaktio)

- Tietokantatapahtumalla tarkoitetaan yhtenä jakamattomana kokonaisuutena pidettävää tietokantaoperaatioiden joukkoa, esimerkiksi tilisiirto:

```
update tili set saldo=saldo-500
   where tilinumero=123456;
update tili set saldo=saldo+500
   where tilinumero=654321;
```

SQL - Tietokantatapahtuma (transaktio)

- Tkhj takaa, että
  - tapahtuma suoritetaan kokonaan eikä vain osaa siitä (ei siis vain tililtäottoa)
  - ulkopuoliset näkevät vain kokonaisen tapahtuman aiheuttamat muutokset (ulkopuolinen ei voi nähdä tilannetta, jossa tililtä 123456 on otettu 500 mutta tilille 654321 ei sitä ole vielä viety)
  - tapahtuman suorituksen aikana tehdyt muutokset kantaan on peruttavissa siihen asti kunnes tapahtumaan on sitouduttu
  - kun tapahtumaan on sitouduttu (se on valmis) muutokset jäävät pysyviksi ja näkyvät myös muille.

SQL - Tietokantatapahtuma (transaktio)

- Tapahtuma päätetään onnistuneesti komennolla **commit [work]**
- Tapahtuma voidaan päättää myös perumalla sen aikaansaamat muutokset komennolla **rollback [work]**
- Tilisiirtotapahtuma olisi kokonaisuudessaan siis

```
update tili set saldo=saldo-500
   where tilinumero=123456;
update tili set saldo=saldo+500
   where tilinumero=654321;
commit;
```

SQL - Tietokantatapahtuma (transaktio)

- Järjestelmät voidaan määritellä toimimaan auto-commit tilassa, jolloin jokaiseen ylläpito-operaatioon sitoudutaan välittömästi (tällöin tilisiirtoa ei voida koota transaktioksi)
- Normaalitilassa tapahtumia kuitenkin kootaan commit operaatioiden avulla. Kahden commitin välissä olevat operaatiot muodostavat tapahtuman.

SQL - Tietokantatapahtuma (transaktio)

- Yritetään poistaa tyhjät harjoitusryhmät, oletetaan, että ilmoittautumisten viiteavaimeen liittyy on delete cascade -määre

```
commit;
select count(*) from ilmoittautumiset;
>> 3500 <<
delete from harjoitusryhma
where ryhmanro is not null;
select count(*) from ilmoittautumiset;
>>> 0 <<<
rollback;
select count(*) from ilmoittautumiset;
 >> 3500 <<
```

Ei ole ihan oikein

(oho!)

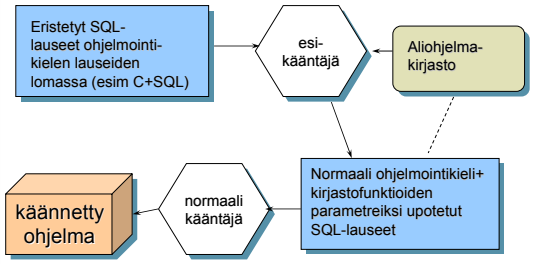
Tietokantaohjelmointi

- Tietokantaa käytetään harvoin suoraan kyselyliittymän kautta
- Tyypillisesti käyttö tapahtuu sovellusohjelman kautta
- Sovellusohjelmaa laadittaessa vaihtoehtoja tietokantakäsitettelyn toteutukseen ovat:
  - Sulautettu SQL (embedded SQL)
  - Ohjelmointirajapinnan (API) kautta tapahtuva käyttö
  - Tietokannan piilottavat välikerrosohjelmistot
  - Erityinen tietokantaohjelmointikieli

Sulautettu SQL

- SQL-lauseet kirjoitetaan ohjelmoitikielen lauseiden joukoon erityisesti **merkittyinä** siten, että esikääntäjä (SQL pre-compiler) osaa tunnistaa ne ja muuntaa ne kirjasto-operaatioiden kutsuiksi
- Esikääntäjän tuottama tulostiedosto käännetään normaalikääntäjällä
- 2-vaiheinen käännös
- Esikääntäjiä saatavissa muutamille kielille, tkhj:n toimittajalta (C, Cobol, Pascal,…, Java)

Esikääntäjän käyttö sulautetussa SQL:ssä



Esimerkki sulautetusta SQL:stä - Pascal

```
function keskipalkka(dept:integer):real;
var
  n: integer;
  psumma: integer;
  #include SQLCA.INC
  EXEC SQL BEGIN DECLARE SECTION
      var  palkka: integer;
           os: integer;
  EXEC SQL END DECLARE SECTION
```

Esimerkki sulautetusta SQL:stä - Pascal

```
begin
   EXEC SQL DECLARE pal CURSOR FOR
      SELECT salary  from employee
      where department= :os;
   n:=0; psumma:=0; os:= dept;
   EXEC SQL  open pal;
   EXEC SQL fetch pal into :palkka;
   while sqlcode = 0 do begin
      psumma := psumma + palkka;
      n := n + 1;
      EXEC SQL fetch pal into :palkka;
   end;
    EXEC SQL close pal;
   if n > 0 then  keskipalkka := psumma/n
   else  keskipalkka := 0;
end;.
```

Sulautetun SQL:n käsitteitä

- Kursori
  - Kyselyn vastausrivijoukon läpikäyntiin käytettävä rakenne
    - määritellään (declare)
    - avataan (open) = kysely suoritetaan avausheykellä voimassaolevilla muuttuja-arvoilla
    - haetaan rivi (fetch) = edetään tulosrivijoukossa + siirretään vuoroon tulevan rivin data ohjelmamuuttujiin
    - suljetaan (close)

Sulautetun SQL:n käsitteitä

- Tietokantaoperaatio voi epäonnistua. Jokaisen tietokantaoperaation jälkeen on tutkittava onnistuiko operaatio
- sqlstate ja vanhemman standardin mukaisesti sqlcode muuttujat palauttavat virhekoodin
- (sqlcode=0, jos kaikki OK, muut arvot erilaisia virhekoodeja - arvot järjestelmäkohtaisia)

Rajapintakirjaston kautta tapahtuva käyttö

- Ohjelmointirajapinnan (API) kautta tapahtuva käyttö perustuu rajapinnan toteuttavan kirjaston käyttöön
- Toimittajakohtaiset kirjastot **Native API**
  - esim OracleCLI = Oracle Call Level Interface
- Toimittajariippumattomat kirjastot
  - esim ODBC (Microsoft Open Database Connection), JDBC Java liittymäkirjasto
  - Mahdollistavat tkhj:n vaihdon, ja useita tietokantoja samassa ohjelmassa

Rajapintakirjaston kautta tapahtuva käyttö

- Toimittajariippumaton kirjasto vaatii kuitenkin tkhj-kohtaisen ajurin toimiakseen tietyn tkhj:n kanssa
- ODBC on yleisimmin käytetty liittymäkirjasto
  - Kaikilla merkittävillä toimittajilla on tarjolla tkhj-kohtaiset ODBC-ajurit. C-kieli tyylinen parametrivälitys.
- JDBC:n perusideat samoja kuin ODBC:n
  - Osa ODBC:n detaljeista piilotettu tietokannankäsittelyluokkien sisään, joten käyttö on hieman yksinkertaisempaa.

JDBC

- Java tietokantakytkentä (JDBC) perustuu muutamaan keskeiseen luokkaan:
- DriverManager
  - luokan palvelujen avulla otetaan käyttöön välttämätön tkhj-kohtainen ajuri (erillinen toimittajalta saatava kirjasto) ja muodostetaan yhteys tietokantaan,
  - Ajurit osaavat yleensä rekisteröidä itsensä, joten ajuriluokan lataus muistiin riittää
  - Tässä kuitenkin rekisteröintikoodi, joka ottaa käyttöön Oracle thin-ajurin Oracle-kantaa varten
  - DriverManager.registerdriver(
    new oracle.jdbc.driver.OracleDriver());

JDBC

- Connection
  - tietokantayhteys - tietokantaistunto
  - yhteys ohjelman ja tietokannan välillä
  - peruspalvelut tietokantatapahtumien käsittelyyn ja tietokantaoperaatioiden muodostukseen
  - kaikki käsittely perustuu yhteyden olemassaoloon ja tapahtuu sen kautta
  - yhteys tulisi lopettaa close operaatiolla kun sitä ei enää tarvita - vapauttaa resursseja
  - DriverManager tarjoaa palvelun yhteyden luontiin

JDBC

```
Connection con =
    DriverManager.getConnection(
      "jdbc:oracle:thin:@kontti.helsinki.fi:1521:ttst",
      "info","expert");
```

Luo yhteyden oracle thin ajuria käyttäen portin 1521 kautta koneessa kontti.helsinki.fi olevaan ttst-nimiseen tietokantaan käyttäen käyttäjätunnusta info ja salasanaa expert.

JDBC

- Statement
  - Mekamismi operaatioiden välittämiseen tietokannanhallintajärjestelmälle ja vastausten palauttamiseen takaisin ohjelmalle
  - palveluja mm.
  - executeQuery - kyselyjen suoritukseen
  - executeUpdate - muihin operaatioihin
- Connection tarjoaa palvelun Statement olion luontiin

JDBC

- ResultSet
  - Kyselyn vastaukset ja niiden käsittely
  - Vastaa sulautetun SQL:n kursori käsitettä
  - Statement.executeQuery luo ResultSet olion
    - operaatiolle annetaan kysely merkkijonoparametrina

```
Statement stmt= con.createStatement();
ResultSet rs= stmt.executeQuery(
   "select nimi, osoite, palkka from henkilo");
```

JDBC

- Vastauksen käsittely ResultSet:n metodeilla
  - Totuusarvoinenfunktio next() aktivoi vastauksen seuraavan rivin. Funktio saa arvokseen true, jos tällainen rivi on olemassa. Ensimmäisellä kutsukerralla aktivoituu ensimmäinen rivi.
  - Perinteisesti vastausjoukkoa on voinut käydä läpi vain yhteen suuntaan: alusta loppuun – uudemmissa versioissa on myös muita mahdollisuuksia

JDBC

- Tiedon saamiseksi aktivoidulta vastausriviltä ohjelman käyttöön on tarjolla tietotyyppikohtaiset hakufunktiot getTyyppi
  - esim. getString, getBoolean, getInt, getDate,….
- Näille funktioille annetaan parametrina joko sarakkeen nimi tai sarakkeen järjestysnumero
- esim:
  - *String a= rs.getString("osoite"); // sarake osoite*
  - *Int p= rs.getInt(3); // kolmas sarake*

JDBC

- Hakufunktiot kykenevät tekemään joitakin tietotyyppikonversioita, esim. merkkijonosta kokonaisluvuksi (jos kyseessä on kokonaisluku) tai päinvastoin. - Ellei konversio onnistu, aiheutetaan SQLException poikkeus - Sama poikkeus aiheutetaan myös muissa virhetilanteissa
  - Tyhjäarvon testaamista varten on totuusarvoinen funktio wasNull. Tämä on parametroitu kuten get-funktiot.

JDBC

```
Statement stmt= con.createStatement();
ResultSet rs= stmt.executeQuery(
   "select nimi, osoite, palkka from henkilo " +
   " order by nimi");
while (rs.next()) {
   System.out.println(rs.getString(1) +",  "+
      rs.getString(2)+",  "+rs.getString(3));
}
```

tulostaa muotoa :

```
Lahtinen Kalle, Katu 6, 12000
Mäki Manu, Kuja5, 20000
```

JDBC

- Tietokannan sisältöä tai rakennetta muuttavat sql-operaatiot, jotka eivät tuota vastausta suoritetaan **Statement.executeUpdate**-operaatiolla.
- esim.

```
Int muutettujaRiveja=
   stmt.executeUpdate("update henkilo "+
     "set palkka= palkka + 10000 " +
     " where nimi= 'Laine Harri' ");
```

JDBC

- **Parametroidut operaatiot:**
  - Edellä on käsitelty yksinkertaisia tapauksia, joissa operaatio annetaan sellaisenaan suoritusfunktion parametrina. Usein kuitenkin samaa operaatiorunkoa käytetään uudelleen, mutta siten, että parametriarvot muuttuvat - esim. käyttäjältä kysytään henkilön nimi ja sitten kannasta haetaan tiedot tämän perusteella
  - Tähän tarkoitukseen on tarjolla 'parametroitu operaatio' PreparedStatement (Statement luokan aliluokka)

JDBC

```
PreparedStatement pst =
        con.prepareStatement(
            "select nimi,osoite,palkka "+
            "from henkilo "+
            "where nimi like ?" );
```

- Parametrin arvon asetukseen on käytössä tietotyyppikohtaiset asetusmetodit setTyyppi (get-funktioita vastaten)
- Asetusmetodilla on kaksi parametria:
  - SQL-operaation parametrin (kysymysmerkin) järjestysnumero ja
  - tilalle tuleva arvo
  - esim. pst.setString(1,"Möttö%");

JDBC