SQL:n käyttö ohjelmissa

- Miksi vuorovaikutteinen käyttö ei riitä?
  - kielen hallinta: maallikot?
  - yhdistetään yleiskielen ja tietokantakielen edut, mm.
    - » monimutkaisempi laskenta
    - » tuloksen muotoilu, näytön hallinta
  - SQL: kohteena joukot (taulut)
  - Pascal , Java (ym.): tietue, tietoalkio tms. kerrallaan

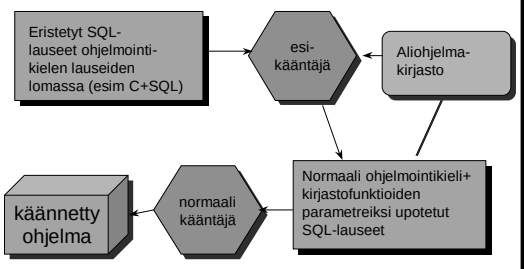
Tietokannan käyttö ohjelmasta

- Tietokannan ohjelmakäytön vaihtoehdot:
  - Sulautettu SQL (embedded SQL) Ohjelmointirajapinnan (API) kautta tapahtuva käyttö
  - 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 tulostiedosto käännetään sitten normaalikääntäjällä
- 2 vaiheinen käännös
- esikääntäjiä muutamille kielille

Sulautettu SQL



Sulautettu SQL - näyttää tältä

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



```
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;.
```

Sulautettu SQL -käsitteitä

- Kursori
  - Kyselyn tulosjoukon läpikäyntiin käytettävä rakenne
    - määritellään (declare)
    - avataan (open) = kysely suoritetaan sen hetkisillä muuttuja-arvoilla
    - haetaan rivi (fetch) = edetään tulosrivijoukossa + siirretään vuorossaolevan rivin data ohjelmamuuttujiin
    - suljetaan (close)

Sulautettu SQL -käsitteitä

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

Käyttö ohjelmointirajapinnan kautta

- 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

Käyttö ohjelmointirajapinnan kautta

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

JDBC:n perusteet

Aika hyvä esitys löytyy kirjasta:

Hamilton g., Cattell R, Fisher M.:
JDBC Database Access with Java -
A tutorial and Annodated Reference,
Addison-Wesley, 1997

- Kirjan esimerkit löytyvät pakattuna osoitteesta
  - http://www.javasoft.com/products/jdbc/book.html ja purettuna Informaaatiojärjestelmät/k99 -kurssin kotisivun kautta

JDBC:n perusteet

- Java tietokantakytkentä (JDBC) perustuu muutamaan keskeiseen luokkaan:
- DriverManager
  - tämän luokan palvelujen avulla otetaan käyttöön välttämätön tkhj-kohtainen ajuri (erillinen toimittajalta saatava kirjasto) ja muodostetaan yhteys tietokantaan.
  - Esim: DriverManager.registerdriver(
    new oracle.jdbc.driver.OracleDriver());

ottaa käyttöön oracle thin-ajurin Oracle-kantaa varten

JDBC:n perusteet

- 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 luo yhteyden

jdbc

```
Connection con =
    DriverManager.getConnection(
      "jdbc:oracle:thin:@kontti.helsinki.fi:1521:tktl",
      "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
  - executeUpdate
- Tietokantayhteys luo Statement olion

jdbc

- ResultSet
  - Kyselyn vastaukset ja niiden käsittely
  - 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
  - Totuusarvofunktio next() aktivoi vastauksen seuraavan rivin. Funktio saa arvokseen true, jos tällainen rivi on olemassa. Ensimmäisellä kutsukerralla aktivoituu ensimmäinen rivi.
  - Huom. next -funktiolla vastausta voi käydä läpi vain yhteen suuntaan. Muita läpikäyntitapoja ei ole (näin versiossa 1.0)

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.

```
stmt.executeUpdate("update henkilo "+
   "set palkka= palkka + 1000000 " +
   " 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 ?" );
```

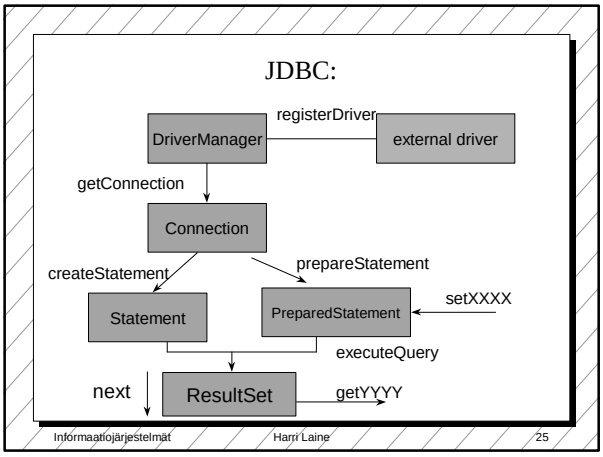
parametri osoitetaan kysymysmerkillä

- 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

- Myös ns. dynaamisen SQL:n käyttö on mahdollista. Tällöin ohjelmoija ei esimerkiksi tiedä minkä tyyppisiä ja kuinka monta saraketta kyselystä tulee vastaukseksi.
- Asian selvittämiseksi voidaan vastausjoukolta kysyä ns. metatietoa ja käyttää sitä sitten apuna vastauksen purkamisessa.

ResultSetMetadata




```
// Esimerkkiohjelma - Informaatiojärjestelmät
// Solid tietokannan käyttö
// HY/TKTL:n Linux ympäristössä tarvitaan CLASSPATH:iin
// "/opt/solid/lib/SolidDriver.zip" , ajurin löytämiseksi
// Harri Laine 16.2.1999

import java.sql.*;
import java.io.*;

public class Esim1  {
  public static void main(String args[]) throws Exception {

  String url = "jdbc:solid://db.cs.helsinki.fi:1414";
  // Tämä url identifioi esimerkkitietokannan, joka toimii koneessa
  // db.cs.helsinki.fi
  Connection con;
  String query = "select KOODI, KURSSI.NIMI AS KNI, OPINTOVIIKOT, "+
                 "OPETTAJA.NIMI AS OPNI " +
                      "from KURSSI, OPETTAJA " +
                      "where KURSSI.LUENNOIJA= OPETTAJA.OPETUNNUS " +
                      "order by OPETTAJA.NIMI, KURSSI.NIMI";
  // Kysely kohdistuu laskuharjoituksissa käytettyyn kantaan
  Statement stmt;
```



```
  try {
    DriverManager.registerDriver(
       (Driver) Class.forName("solid.jdbc.SolidDriver").newInstance());
  } catch(java.lang.ClassNotFoundException e) {
      // Ilmoitetaan siitä, ettei yhteyttä saada aikaan.
          System.err.print("ClassNotFoundException: ");
          System.err.println(e.getMessage());
   }

  try {
          // Luodaan yhteys kurssin käyttäjätunnuksella
      con = DriverManager.getConnection(url, "info", "expert");
          // Luodaan tietokantaoperaation suoritusympäristö
      stmt = con.createStatement();
          // Muuttujaa ope käytetään pitämään kirjaa edellisestä
          // opettajasta. Tällä kontrolloidaan opettajan nimen
          // tulostusta.
      String ope =" ";
          // Tulostetaan otsake
      System.out.println("OPETUSTEHTÄVÄT:");
      System.out.println();
          // Suoritetaan kysely
      ResultSet rs = stmt.executeQuery(query);
```



```
        // Käydään läpi vastausrivit
    while (rs.next()) {
          // Eristetään opettajan nimi, huom. alias-nimen käyttö
      String opn = rs.getString("OPNI");
          // Jos opettaja vaihtuu tulostetaan tyhjä rivi ja
          // opettajan nimi, muuten vain kurssin tiedot
      if (ope.equals(opn)) {
        System.out.println("     " +
          rs.getString("KNI")+ " (" + rs.getInt("KOODI")+")");
      }
      else {
        System.out.println();
        ope= opn;
        System.out.println(ope);
        System.out.println("     " +
          rs.getString("KNI")+" ("+rs.getInt("KOODI")+")");
      }
    }
```



```
      // Suljetaan 'kursori' - vapautetan vastausjoukon käsittelyyn
      // kiinnitetyt resurssit. Tälle vastausjoukolle ei enää voi
      // tehdä mitään, vaikka sitä ei suljettaisikaan.
    stmt.close();
      // Suljetaan tietokantayhteys. Yhteys suljetaan vasta siinä
      // vaiheessa kun ohjelmassa ei enää käsitellä kantaa.
    con.close();
  } catch(SQLException ex) {
      // Imoitetaan virheestä
          System.err.print("SQLException: ");
          System.err.println(ex.getMessage());
    }
  }
}
```