Teema10. Protseduuride keel SPL (Stored Procedure Language), protseduurid, veatöötlus SQL-is. Trigerid.

Protseduuride keel SPL
(Stored Procedure language)

Ei ole standardi osa. SPL ei ole SQL. SQL ei ole protseduurne keel, seal puuduvad võimalused programmi struktuuri kirjutamiseks (valik, kordus). Need võimalused on SPL-is. SPL on SQL-i laiendus, temas saab kasutada SQL-i käske ja struktuurse programmeerimise vahendeid. SPL-i kasutatakse selleks, et kirjutada salvestatud protseduure (Stored procedure). Salvestatud protseduur on SPL keeles kirjutatud programm (funktsioon), mis on täidetaval (kompileeritud) kujul salvestatud andmebaasi (seega mitte eraldi teksti- vms. faili, vaid andmebaasi tabelitesse).
Omadused:
1) Salvestatud protseduurid on andmebaasis kompileeritud kujul - seega jääb ära süntaksikontroll ja kompileerimine ning sageli kasutatavate SQL-lausete puhul on ajavõit suur
2) SPL protseduur on andmebaasi objekt - seega on kättesaadav igale andmebaasi kasutajale ja paljudele rakendustele. Mitu rakendust võivad kasutada samu protseduure.

Salvestatud protseduuri loomine (see on SQL lause)

```
CREATE PROCEDURE proc_name
( [ var_name {type |LIKE table.column} [DEFAULT expr]
  [, var_name {type |LIKE table.column} [DEFAULT expr]]... )
  [RETURNING type [,type]...;]
statement_block
END PROCEDURE;
```

Protseduuri nime järgi on sulgudes parameetrite loetelu koos tüüpidega ja võimalike vaikeväärtustega. Viimast kasutatakse juhul, kui protseduuri väljakutsumisel on argumente vähem kui protseduuri kirjelduses argumente.

Näiteks:

```
CREATE PROCEDURE a () RETURNING int;
   DEFINE x INT;
   LET x=20;
   RETURN x;
END PROCEDURE

CREATE PROCEDURE leiatud (tudvoti CHAR(10))
   RETURNING CHAR(20),CHAR(15),INTEGER;
DEFINE perek CHAR(20);
DEFINE eesnimi CHAR(15);
DEFINE kurs INTEGER;

SELECT pnimi,enimi,kursus
   INTO perek,eesnimi,kurs FROM tudeng
   WHERE tkood = tudvoti;
RETURN perek,eesnimi,kurs;
END PROCEDURE;
```

Protseduuri väljakutse:

1) SQL keele abil
EXECUTE PROCEDURE proc_name
  ([[par_name =] expr [,[par_name=] expr]...])
  [INTO var [,var]...]

INTO-klausel on môeldud ainult SPL-s täitmiseks, sinna muutujatesse tulevad tagastatavad väärtused.
Näiteks:
EXECUTE PROCEDURE leiatud (1234)
  INTO pn,en,kr;

2) SPL-is ainult
CALL proc_name
  ([[par_name =] expr [,[par_name=] expr]...])
  [RETURNING var [,var]...]
Näit.
CALL leiatud (1234) RETURNING pn,en,kr;

3) SPL-is funktsioonina avaldises:
Näiteks:
LET pn,en,kr = leiatud(1234)

Protseduurikeel
Kommentaarid
-- see on kommentaar

Muutujad
SQL keele lausetes saab muutujat kasutada igal pool, kus konstanti vôi avaldist. SPL - is omistamine ja avaldised. Muutuja nime puhul kehtivad samad kitsendused kui teiste nimedegi puhul. Muutujad tuleb defineerida. Selleks on kaks vôimalust:
1. Otsene defineerimine DEFINE-lausega:
DEFINE variable [,variable]... type;
DEFINE x,y SMALLINT;
2. Kaudne defineerimine protseduuri päises:
CREATE PROCEDURE ruut (x SMALLINT, y SMALLINT)
  RETURNING integer;
- see päis defineerib muutujad x ja y

Kui muutujal ja tabeli väljal on üks ja sama nimi, siis vaikimisi eeldatakse muutujat. Kui on vaja viidata tabeli väljale, tuleb sel juhul lisada ette tabeli nimi.

Omistamine
Enne muutuja kasutamist tuleb talle väärtus omistada (see vôib olla ka NULL)!
Neli vôimalikku viisi:
1) LET-lause:
LET a = x+y;
LET a,b = c,d;
LET pn,en= (SELECT pnimi,enimi FROM tudeng WHERE tkood=1234)
2) SELECT ... INTO
SELECT pnimi,enimi INTO pn,en FROM tudeng WHERE tkood=1234
3) CALL ... RETURNING
Näiteks: CALL leiatud (1234) RETURNING pn,en,kr;
4) EXECUTE PROCEDURE ... INTO
EXECUTE PROCEDURE leiatud (1234) INTO pn,en,kr;

Valik
Traditsiooniline IF ja CASE on ühes lauses:

```
IF a>b THEN
  LET x=1;
ELIF a>c THEN
  LET x=2;
ELSE
  LET x=0;
END IF
```

Kordus
Kolm võimalust alustada kordust:
1) FOR - alustab kindel arv kordi läbitava tsükli. Lõpetamine on garanteeritud

```
FOR i=1 TO 10 STEP 2
 ...
END FOR
FOR x IN ('Jaan','Jüri',
  (SELECT enimi FROM tudeng WHERE tkood=1234))
  DELETE FROM tudeng WHERE enimi=x;
END FOR
```

2) FOREACH - lubab selekteerida (SELECT) ja töödelda rohkem kui ühte rida andmebaasist.

```
FOREACH SELECT tkood INTO x FROM tudeng WHERE suund='LA'
  INSERT INTO oppim (tudkood,akood,regkuup)
    VALUES (x,LDU4430,'14.03.1995');
END FOREACH
```

3) WHILE - tingimuslik kordus. Lõpetamine ei ole garanteeritud

```
WHILE condition
 statement_block
END WHILE
```

Kordust täidetakse seni kuni tingimus on tõene.

Neli võimalust lõpetada kordust:
1) CONTINUE - jätab täitmata tahapoole jäävad korduse laused ja alustab uut iteratsiooni:

```
CONTINUE { FOR | WHILE | FOREACH }
LET a=0;
FOR i=1 TO 10
  IF a>5 THEN
    CONTINUE FOR;
  END IF
  LET a = a+1;
END FOR
```

2) EXIT - Väljub kordusest ja jätkab protseduuri täitmist esimesest korduse lõpule järgnevast reast

```
EXIT { FOR | WHILE | FOREACH }
WHILE i<10
  LET i = i+1;
  IF i>6 THEN
    EXIT WHILE
  END IF
END WHILE
```

3) RETURN - väljub tervest protseduurist

```
FOR i =1 TO 10
```

```
  LET x = i
  IF i=4 THEN
    RETURN x
  END IF
END FOR
```

4) RAISE EXEPTION
Simuleerib vea tekkimist

Lausebloki struktuur:

Statement_block:
[DEFINE statements]
[ON EXCEPTION statements]
statements

Lubatud laused:
CALL, CONTINUE, EXECUTE PROCEDURE, EXIT, FOR, FOREACH, IF, LET, RAISE EXCEPTION, RETURN, WHILE, SYSTEM, WHILE, SQL-lause, BEGIN statement_block END

SQL-keelest võib kasutada kõiki, v.a.
CREATE PROCEDURE, LOAD, UNLOAD, jms.

FOREACH tsükkel

FOREACH-tsükkel om mõeldud mingi hulga andmebaasi tabeli kirjete töötlemiseks tsükli sees. Tavalise SQL keele juures ei saa nende lausete puhul, mille tulemuseks on hulk kirjeid, iga kirjega eraldi midagi ette võtta. Kasutades SPL-i FOREACH-tsüklit on see võimalik.
Uus mõiste - kursor - võib vaadelda kui ajutiselt väljavalitud (aktiviseeritud) osa andmebaasi tabelist, mille SPL (SQL) loob kas SELECT-lausest või sellisest EXECUTE PROCEDURE lausest, mis tagastab palju ridu (selles peab siis järelikult olema SELECT-lause). Tavaliselt luuakse kursor ainult ühekordseks läbimiseks, kuid on võimalik ka mitmeid kordi läbida. Interaktiivses SQL-is (dbaccess Informixis, sqlplus Oracle-s) tavaliselt kursoreid ei kasutata, nende peamine kasutusvaldkond on Embedded SQL -keeled (näiteks C, Fortran, COBOL), kus kursorite abil toimub andmevahetus programmi muutujate ja andmebaasi tabelite vahel.

Süntaks (lihtsustatud):
1) FOREACH [cursor_name FOR] SELECT-INTO-statement
    statement_block
  END FOREACH;
2) FOREACH EXECUTE PROCEDURE proc_name
  ([[par_name =] expr [,[par_name=] expr]...])
  [INTO var [,var]...]
    statement_block
  END FOREACH;

Näiteks - kõik 5. kursuse tudengid lopetavad ja tuleb tabelitest eemaldada nende oppimised ja tudengi enda andmed:

```
DEFINE dipl CHAR(10);
FOREACH SELECT tkood INTO dipl FROM tudeng
  WHERE kursus=5
-- nüüd on muutujas kustu järjekordse 5.kurs. tudengi kood
  DELETE FROM oppim WHERE tudkood = dipl;
```

```
  DELETE FROM tudeng WHERE tkood = dipl;
END FOREACH;
```

Vôi oletame, et meil on protseduur, mis leiab parameetriga antud kursuse tudengite koodid:

```
CREATE PROCEDURE sama_kurs (aasta INT)
  RETURNING CHAR(10);
  DEFINE kood CHAR(10);
  SELECT tkood INTO kood FROM tudeng WHERE kursus = aasta;
  RETURN kood;
END PROCEDURE;
```

Siis vôiks sama töö teha ära järgmine tsükkel:

```
DEFINE dipl CHAR(10);
FOREACH EXECUTE PROCEDURE sama_kurs (5) INTO dipl;
  DELETE FROM oppim WHERE tudkood = dipl;
  DELETE FROM tudeng WHERE tkood = dipl;
END FOREACH;
```

Sama töö kursori kasutamisega:

```
DEFINE dipl CHAR(10);
FOREACH kursor1 FOR SELECT tkood INTO dipl FROM tudeng
  WHERE kursus=5
-- nüüd on muutujas kustu järjekordse 5.kurs. tudengi kood
  DELETE FROM oppim WHERE tudkood = dipl;
  DELETE FROM tudeng WHERE CURRENT OF kursor1;
END FOREACH;
```

Kursori kasutamine on kiirem, kuna andmebaasil ei ole vaja enam vastavat kirjet otsida, see on kohe olemas kursoris.

Kursorite kasutamiseks andmebaasi uuenduses (kustutamine ja parandamine) on eraldi süntaks:

```
DELETE FROM table WHERE CURRENT OF cursor_name;
UPDATE table SET ... WHERE CURRENT OF cursor_name;
```

ÜLESANNE: täiendada eelpooltoodud programmilôiku nii, et see
muutuks semestri lôpetamise protseduuriks, mis
1) jätab endisse seisu kôik need, kellel on môni eksam tegemata vôi on selle hinne 1
2) kustutab andmebaasist 5. kursuse edukate tudengite andmed
3) viib ülejäänud edukad tudengid järgmisele kursusele ja kustutab nende oppimiste andmed

```
CREATE PROCEDURE sem_lopp ()
DEFINE jtud CHAR(10);
DEFINE kurs INT;
FOREACH kursor1 FOR SELECT tkood,kursus
                  INTO jtud,kurs FROM tudeng
  -- tsükkel algab
  IF EXISTS (SELECT tudkood FROM oppim WHERE tudkood=jtud
    AND (hinne IS NULL OR hinne = 1)) THEN
    -- need on mitteedukad
    CONTINUE FOREACH;
  ELIF kurs = 5 THEN
    DELETE FROM oppim WHERE tudkood = jtud;
    DELETE FROM tudeng WHERE CURRENT OF kursor1;
  ELSE
    DELETE FROM oppim WHERE tudkood = jtud;
    UPDATE tudeng SET kursus = kursus+1
```

```
      WHERE CURRENT OF kursor1;
  END IF
END FOREACH;

END PROCEDURE;
```

Veatöötlus

SQL ja ISAM vigade püüdmiseks ning töötlemiseks:

```
ON EXCEPTION [IN (errnum [,errnum]...)]
  [ SET SQL_err_var [, ISAM_err_var [, err_data_var ]]]
  statement_block
END EXCEPTION [WITH RESUME] [;]
```

Kui on antud veanumbrite loetelu (errnum), siis püütakse kinni ainult lotletud numbritega vead, vastasel puhul kôik.
Kui SET-klausel on antud, siis salvestatakse pärast vea tekkimist vastavatesse muutujatesse SQL vea number, ISAM vea number ja SQL vea tekst.
WITH RESUME-klausel - pärast vea töötlemist jätkatakse protseduuri täitmist viga pôhjustanud lausele järgnevast lausest.
NB! ON EXCEPTION on deklaratiivne lause, temas sisalduvaid käske ei täideta mitte protseduuri täitmise järjekorras, vaid alles siis, kui vastav viga tekib. Kui viga ei teki, siis neid lauseid ei täidetagi.

Näiteks protseduur tudengi lisamiseks, mis kontrollib, kas tudengi tabel ikka on loodud. Kui ei ole, siis loob selle enne lisamist:

```
CREATE PROCEDURE lisa_tud (kood CHAR(10),pn CHAR(20),
  en CHAR(15), kr integer DEFAULT 1) RETURNING INT;
  DEFINE oliviga INT;
  ON EXCEPTION IN (-206)  -- Viga - tabelit ei ole
    CREATE TABLE tudeng (tkood CHAR(10),...);
    INSERT INTO tudeng (tkood,pnimi,enimi,kursus)
      VALUES (kood,pn,en,kr);
    LET oliviga = 1;
  END EXCEPTION WITH RESUME;
  LET oliviga = 0;
  INSERT INTO tudeng (tkood,pnimi,enimi,kursus)
    VALUES (kood,pn,en,kr);
  RETURN oliviga;
END PROCEDURE;
```

Vôib püüda ka mitmeid vigu korraga:

```
...
DEFINE sql_viga,isam_viga INT;
DEFINE veatekst CHAR(50);
ON EXCEPTION SET sql_viga,isam_viga,veatekst
  ...
  IF sql_viga = -206 THEN
    ...
  END IF
...
END EXCEPTION;
```

Vea kunstlikuks tekitamiseks (simuleerimiseks):

RAISE EXCEPTION SQL_err [, ISAM_err [, err_text]];

RAISE EXCEPTION lause täitmisel lõpetatakse protseduuri täitmine, kui enne pole kirjeldatud selle vea jaoks ON EXCEPTION -it.

Näiteks:

```
CREATE PROCEDURE lisa_tud (kood CHAR(10),pn CHAR(20),
  en CHAR(15), kr integer DEFAULT 1)
  IF kr > 5 THEN
    RAISE EXCEPTION -99999,0,'Liiga suur kursus!';
  END IF
  INSERT INTO tudeng (tkood,pnimi,enimi,kursus)
    VALUES (kood,pn,en,kr);
END PROCEDURE;
```

- **Andmebaasi trigerid.**

(Oracle- näitel)

Trigerid (*Triggers*) on põhimõtteliselt andmebaasi kompileeritud kujul salvestatud protseduurid, mis on seotud mingi konkreetse andmebaasi tabeliga (või tabeli mingite väljadega) ning mis käivutuvad automaatselt siis, kui selle tabeli kirjeid muudetakse (lisatakse, muudetakse, kustutatakse), seega kui toimub mingi andmetabeli sisu puudutav sündmus. Ei ole veel (?) SQL-I standardis, kuid praktiliselt kõik andmebaasisüsteemid pakuvad seda võimalust. Kuna pole standardiseeritud, erineb trigerite süntaks ja kasutamise võimalused erinevates süsteemides mõnevõrra.

Trigereid on kolme põhiliiki - INSERT- , UPDATE ja DELETE- trigereid, liigitus vastava sündmuse järgi, millele triger reageerib.

INSERT - triger käivitub siis, kui andmebaasi lisatakse uus kirje. On olemas kaks alaliiki - BEFORE INSERT ja AFTER INSERT trigerid.
BEFORE INSERT - käivitatakse enne uue kirje lisamist tabelisse.
AFTER INSERT - käivitatakse pärast kirje lisamist tabelisse.

UPDATE - triger käivitub siis, kui olemasolevat kirjet tabelis muudetakse.(UPDATE lause). Analoogselt INSERT-trigeriga olemas BEFORE UPDATE ja AFTER UPDATE trigerid.

DELETE - triger käivitub kirje kustutamisel tabelist. Olemas BEFORE DELETE ja AFTER DELETE trigerid.

Trigeri süntaks - koosneb

1. Päisest, kus määratakse ära, mis liiki trigeriga (INSERT, UPDATE, DELETE) on tegemist ja millise tabeliga on see triger seotud.Triger võib olla seotud ainult ühe

tabeliga. Oracle puhul on nii, et UPDATE- triger võib olla kirjutatud ka ainult mingi kindla veeru muutmise kohta kirjes (mitte terve kirje kohta)
2. kehast (*trigger body*) - koosneb SPL- keele lausetest ja SQL lausetest. Need laused täidetakse siis, kui toimub vastav sündmus (*triggering statemant*) mis käivitab trigeri.

Lihtsama trigeri näide:

```
CREATE TRIGGER dummy

BEFORE DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
WHEN (new.empno > 0)
DECLARE
  /* variables, constants, cursors, etc. */
BEGIN
  /* PL/SQL block */
END;
/
```

```
CREATE TRIGGER uus

BEFORE UPDATE ON arve
REFERENCING new AS uusarve
FOR EACH ROW
BEGIN
  :uusarve.kpv := SYSDATE;
END;
```

Võtmesõnad trigeris :
FOR EACH ROW - tähendab, et trigeri käivitumisel rakendatakse trigeri kehas sisalduvaid laused kõikidele tabeli kirjetele.
REFERENCING new AS - trigeris on võimalik viidata nii vanale (veel muutmata) kirjele (old) ja selle veergudele ning uuele, pärast muutmist tekkivale kirjele (new). Sõltumata sellest, millise trigeriga on tegemist (kas BEFORE- või AFTER- tüüpi trigeriga) on viited uuele ja vanale kirjele võimalikud. Kui n. BEFORE INSERT trigeris on viidatud uuele kirjele ja selle kirje mingile väljale on trigeri kehas oleva algoritmiga antud mingi väärtus, siis salvestatakse kirjesse see trigeris antud väärtus, seda isegi juhul, kui kirje lisamisel oli kirjes selle välja jaoks juba väärtus olemas (see nö. kirjutatakse üle trigeri poolt).
Viidetele uue ja vana kirje jaoks võib anda oma nime. (vaikimisi **old** ja **new**)

Näiteks:
CREATE TRIGGER uus

```
BEFORE UPDATE ON palk
REFERENCING old AS vanapalk
FOR EACH ROW
BEGIN
  IF :new.summa < :vanapalk.summa THEN :new.summa = :vanapalk.summa ;
END;
```

See triger ei lase tabelis **palk** kirje muutmisel palka vähendada. Kui uus palgasumma on väiksem kui summa vanas, muutmata kirjes, jääb summa muutmata (e. muudatud kirje palgasummaks saab vana kirje palgasumma)

Trigerite kasutamise reeglid:

- **ühele tabelile saab korraga kirjutada ainult ühe sama tüüpi trigeri** (seega siis Oracle's kuus trigerit - INSERT, UPDATE, DELETE trigerid , mõlemaid kaks AFTER ja BEFORE tüüpi)
- **trigereid tuleks kasutada tsentraalsete, kasutajaprogrammidest sõltumatute protseduuride täitmiseks, selliste protseduuride jaoks, mis tingitud ainult antud andmetabeli andmete muutmisestja mis peavad saama läbiviidud sõltumata sellest, milline kasutaja või kasutajaprogramm seda tabelit parajasti kasutab. Ning kui selline andmete muutmisest tulenev protseduur on vajalik, kasutada selleks eelistatavalt trigerit, mitte kasutajaprogrammi.**
- **trigerite kasutamisel tuleks jälgida seda, et ei dubleeritaks juba omadusi ja võimalusi, mis on andmebaasisüsteemi (n . Oracle'sse) sisse ehitatud.** Näiteks ei ole mõtet kirjutada trigerit, mis kirje kustutamisel kustutaks ära ka seotud kirjed, selle jaoks tuleks kasutada välisvõtmete deklareerimist (nn. deklaratiivne terviklikkus), REFERENCE klauslit tabelite loomisel koos CASCADE ON DELETE -klausliga, mis automaatselt kindlustab seotud kirjete kustutamise **Oracle's.**
- **Trigeritest saab käivitada protseduure.** Kui trigeri algoritm läheb liiga suureks või keerukaks, on trigeri keha mõttekas (võimaluse korral) vormistda eraldi protseduurina ja kasutada trigeri kehas EXEC PROCEDURE lauset.
- **Trigeriga võib muuta teiste andmetabelite sisu**. Siin tuleb olla hoolikas, et mitte tekitada rekursiivseid trigereid - näiteks kui kirjutada AFTER UPDATE triger TOOTAJA tabeli jaoks, mis omakorda lisab uue kirje tabelisse TOOTAJA, siis on tegemist rekursiivse trigeriga, mis jääbki käima.
- Eelnevale punktile lisaks - **mutating tables** - selline väljend käib nende tabelite kohta , mida parajasti muudab see SQL- lause, mis trigeri käivitab. Trigeri abil ei saa enamasti muuta (lugeda, lisada , muuta, kustutada) neid nn. mutating tabeleid, st. triger ei saa muuta tabelid, mille muutmine selle trigeri enda käivitas. (siin on mõningaid erandeid, näiteks BEFORE INSERT trigerist saab muuta ja lugeda seda tabelt, miilele antud triger on kirjutatud)
- **Kõiki andmebaasi terviklikkuse ja andmete õigsuse reegleid ei saa kindlustada deklaratiivse terviklikkusega** (tabelite ja piirangute defineerimisega). Keerulisemate

terviklikkuse reeglite korral ja muudel juhtudel [kui näiteks uue kirje lisamisel on selle kirje õigsust kontrollida , kasutades paljusid erinevaid andmetabeleid, kui on vaja arvutada kirjele lisaatribuute serveri poolt (n. panna arve kuupäevi), kui kirje lisamisel , muutmisel , kustutamisel on vaja teha mitmeid erinevaid muudatusi teistes andmetabelites (mis võivad asuda teistel serveritel, teistes andmebaasides))] on kõige kindlam kasutada andmebaasi trigereid. Trigerid võimaldavad andmebaasi kasutuses järgida ja kontrollida keerulisi, ettevõtte tegevusreeglitest tulenevaid nn. ärireegleid (*business rules*) Näiteks võib triger kontrollida, et arvutis registreeritud varuosade(kaupade) arv ei langeks alla ettenähtud piiri ja kaupade vähenemisel sell piirini genereerida vastava teate või isegi uue kaubatellimuse.

- **Trigerite kasutamine valede andmeuuenduste vältimiseks ja vastavate veateadete väljastamiseks**.
- **Trigerid on kasutatavad tuletatavate veergude väärtuste arvutamiseks vt. eelnevaid trigerite näiteid.** (n. on vaja panna arvele kuupäevi arve sisestamisel) Eriti kasulik on serveri trigeri kasutamine veeru väärtuste arvutamiseks siis, kui nende väärtuste arvutamiseks tuleb läbi töödelda andmeid mitmes andmetabelis. Kiiruse vahe on märgatav, võrreldes sellega, kui sama töö teeb ära klientarvutil asuv rakendusprogramm.
- **Trigeriga on võimalik teha andmebaasi auditit**, st. jälgida ja salvestada, kes milliseid tabeleid kuidas kasutab (lisab, muudab, kustutab), enamuses süsteemides saab trigeri kehas küsida andmebaasi kasutaja nime, kes käivitas trigeri (trigerit käivitava andmemuutuse).

- **Transaktsioonid**

COMMIT - transaktsiooni lõpetamine (uue algus)
ROLLBACK - transaktsiooni tühistamine (tagasirullimine)
SAVEPOINT - salvestuspunkti seadmine, kuhu on võimalik transaktsiooni töö tagasi rullida
SET TRANSACTION - käimasoleva (current) transaktsiooni omaduste määramine.