Podatkovno procesiranje v relacijskih sistemih

Iztok Savnik, FAMNIT.

Vsebina

- Uvod
- Dinamični SQL
- Shranjene procedure in funkcije
- Prožilci

Uvod

- Začetno so SUPB samo shranjevali podatke in nudili vmesnik do podatkom aplikacijam
  - Aplikacije so same delale s podatki
- Podatkovne baze dobivajo vedno več vlog
  - Referenčna integriteta
  - Validacija podatkov, integriteta podatkov, …
  - Sortiranje, agregacija, grupiranje, …
  - Vedno več podatkovnega procesiranja

Podatkovno procesiranje

- Shranjene procedure, funkcije in prožilci
  - Posledice razvoja relacijskih sistemov
- Shranjene procedure se izvajajo v relacijskem sistemu samem !
  - Ni potrebe po prenosu podatkov
- Funkcije se uporabljajo v SQL stavkih
  - Vrnejo vrednost za vsako n-terico relacije
  - Izračuni in tranformacije podatkov pri prikazu rezultatov poizvedb
  - Množica predefiniranih funkcijje del sistema

Podatkovno procesiranje

- Prožilci omogočajo avtomatično proženje shranjenih procedur na osnovi shranjenih podatkov

Proceduralni SQL

- Sybase SQL server je prvi definiral shranjene procedure (1989)
- Glavna prednost je hitrost podatkovnega procesiranja v okolju klient/strežnik
  - Brez shranjenih procedur se mora vsak SQL stavek poslati na strežnik in rezultat se vrne klientu
  - Zelo primerno orodjev v okolju klient/strežnik
- Drugi proizvajalci
  - Osnova: C in Pascal, ujemanje z SQL, Ada (Oracle)

Dinamični SQL

- Aplikacije kjer program konstruira SQL stavke
  - Procesne PB
  - Senzorske PB, ...
  - Internet stvari
- Vgnezden SQL (ponovitev)
- Dinamični SQL
  - Takojšnje izvajanje
  - Dinamično izvajanje

Vgnezden SQL

Pristop:

Vgnezdi SQL stavke v gostiteljski jezik.
Pred-procesor prevede SQL stavke v posebne API klice.
Potem uporabimo običajen prevajalnik za prevajanje kode .

Gradniki jezika:

Priklučitev na SUPB:
EXEC SQL CONNECT
Deklaracija vrednosti:
EXEC SQL BEGIN (END) DECLARE SECTION
Stavki:
EXEC SQL Statement;

Vgnezden SQL: Spremenljivke

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

Two special “error” variables:

- SQLCODE (long, is negative if an error has occurred)
- SQLSTATE (char[6], predefined codes for common errors)

Kurzorji

Definiramo lahko kurzor na relaciji ali poizvedbi (ki generira relacijo).

Lahko odpremo kurzor, preberemo n-terico in potem premaknemo kurzor naprej za eno mesto. Potem spet preberemo n-terico...
Vrstni red n-teric v relaciji, ki jo beremo je določen z SQL vprašanjem.

Lahko tudi spremenimo ali izbrišemo zapis na katerega kaže kurzor.

Primer: Kurzor, ki poišče imena mornarjev, ki so rezervirali rdečo ladjo, v abecednem vrstnem redu

```
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname
    FROM Sailors S, Boats B, Reserves R
    WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
    ORDER BY S.sname
```

SQL vgnezden v C: Primer

```
char SQLSTATE[6];
 EXEC SQL BEGIN DECLARE SECTION
 char c_sname[20]; short c_minrating; float c_age;
 EXEC SQL END DECLARE SECTION
 c_minrating = random();
 EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
 do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
 } while (SQLSTATE != '02000');
 EXEC SQL CLOSE sinfo;
```

Dinamični SQL

SQL vprašanja so znana v času prevajanja.

Dinamični SQL omogoča konstrukcijo SQL stavkov on-the-fly.

Primer:

```
char c_sqlstring[]=
  {“DELETE FROM Sailors WHERE raiting>5”};
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

Omejitve statičnega SQL

- Delo s kurzorji je statično zakodirano v programu
  - Spremenljivke lahko uporabimo za parametre SQL vprašanja

```
exec sql select name, quota, sales
         from salesreps
         where quota > :cutoff_amount;

exec sql update salesreps
         set quota = quota + :increase
         where quota > :cutoff_amount;
```

Omejitve statičnega SQL

- Ne moremo pa spreminjati imen tabel in stolpcev
  - Poskus izvajanja vrne napako

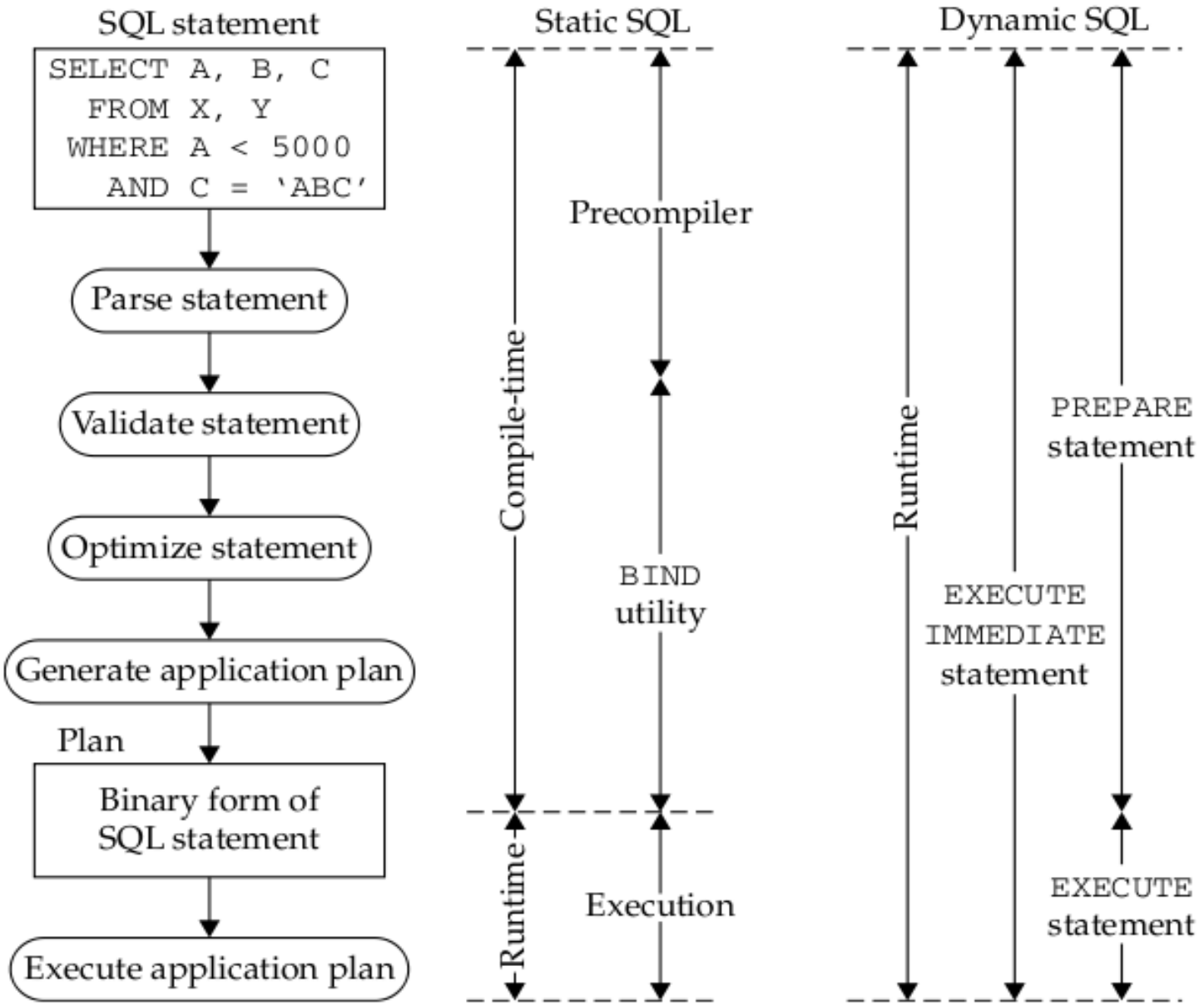
```
exec sql update :which_table
         set :which_column = 0;

exec sql declare cursor cursor7 for
         select *
         from :which_table;
```

- Spremeni se lahko rezultat
  - Stolpci v tabeli, ki je rezultat poizvedbe se spremenijo

Osnove dinamičnega SQL

- Ne zakodiraj SQL stavka v program gostitelja
  - Omogoči gradnjo SQL stavka v programu
- Nekatere enostavne ideje se zakomplicirajo
- Izvajanje SQL iz stališča prevajalnika
  - Statični SQL
    - V času prevajanja se določi program (opt) SQL stavka
    - V času izvajanja se samo interpretira
  - Dinamični SQL
    - SQL stavek ni znan do izvajanja
    - Program SQL stavka se določi v času izvajanja
    - SQL se lahko izgradi v programu gostitelja



PMJ, r-data-proc

Komentarji

- SQL stavki implementirani z dinamičnim SQL se izvajajo počasneje
- Večina programerjev ne uporablja dinamičnega SQL ker ni potrebe
- Aplikacije klient/strežnik velikokrat zahtevajo uporabo dinamičnega SQL

1) Takojšnje izvajanje SQL

- Najenostavnejši način je takojšnje izvajanje SQL stavka iz niza znakov
  - EXECUTE IMMEDIATE
- Koraki:
  - Program konstruira SQL stavek. SQL stavek ne sme vračati vrednosti !
  - Program pošlje SQL stavek z EXECUTE IMMEDIATE SQL strežniku
  - SQL stavek se izvede na SQL strežniku

Zgled:

```
main()
{
   /* This program deletes rows from a user-specified table
      according to a user-specified search condition.
   */

   exec sql include sqlca;
   exec sql begin declare section;
      char stmtbuf[301];              /* SQL text to be executed */
   exec sql end declare section;

   char tblname[101];                 /* table name entered by user */
   char search_cond[101];             /* search condition entered by user */

   /* Start building the DELETE statement in stmtbuf */
   strcpy(stmtbuf, "delete from");

   /* Prompt user for table name; add it to the DELETE statement text */
   printf("Enter table name:      ");
   gets(tblname);
   strcat(stmtbuf, tblname);

   /* Prompt user for search condition; add it to the text */
   printf("Enter search condition:");
   gets(search_cond);
   if (strlen(search_cond) > 0) {
      strcat(stmtbuf, " where ");
      strcat(stmtbuf, search_cond);

   }

   /* Now ask the DBMS to execute the statement */
   exec sql execute immediate :stmtbuf;
   if (sqlca.sqlcode < 0)
      printf("SQL error: %ld\n", sqlca.sqlcode);
   else
      printf("Delete from %s successful.\n", tblname);

   exit();
}
```

Zgled:

- Izvajanje programa

```
Enter table name: staff
Enter search condition: quota < 20000
Delete from staff successful.
```

- Stavek, ki se je izvajal

```
delete from staff
  where quota < 20000
```

- Stavki SQL: DELETE, INSERT, UPDATE, COMMIT

2) Dinamično izvajanje

- Dinamični SQL se izvaja hitreje od EXECUTE IMMEDIATE
- Z dinamičnim SQL izvajamo stavke, ki jih generira uporabnik in se izvajajo zelo velikokrat
- Koraki izvajanja

1) Program pripravi SQL stavek. Vprašaj »?« se vstavi na mesto kjer se pričakuje spremenljivke. Vprašaj imenujemo *vrzel*.
2) Stavek PREPARE zahteva od strežnika, da razčeni, preveri in optimizira SQL stavek. Rezultat je plan izvajanja SQL stavka. SQL strežnik postavi SQLCODE/SQLSTATE.

2) Dinamično izvajanje

- Koraki izvajanja

  3) Ko program želi izvajati pripravljeni SQL stavek naredi to ukazom EXECUTE s katerim se prenesejo tudi parametri. SQL strežnik postavi parametre SQL stavka in izvaja prej pripravljen plan.

  4) Program lahko izvaja stavek EXECUTE večkrat s spremenjenimi parametri.

Zgled

- Splošen program za izvajanje stavka UPDATE
- Začetni stavek (niz znakov)

```
update table-name
   set second-column-name = ?
 where first-column-name = ?
```

- Uporabnik določi parametre pred izvajanjem SQL stavka

Zgled:

```
main()
{
   /* This is a general-purpose update program. It can be used
      for any update where a numeric column is to be updated in
      all rows where a second numeric column has a specified
      value. For example, you can use it to update quotas for
      selected salespeople or to update credit limits for
      selected customers.
   */

   exec sql include sqlca;
   exec sql begin declare section;
      char  stmtbuf[301]          /* SQL text to be executed */
      float search_value;         /* parameter value for searching */
      float new_value;            /* parameter value for update */
   exec sql end declare section;

   char tblname[31];              /* table to be updated */
   char searchcol[31];            /* name of search column */
   char updatecol[31];            /* name of update column */
   char yes_no[31];               /* yes/no response from user */

   /* Prompt user for table name and column name */
   printf("Enter name of table to be updated:    ");
   gets(tblname);
   printf("Enter name of column to be searched:  ");
   gets(searchcol);
   printf("Enter name of column to be updated:   ");
   gets(updatecol);

   /* Build SQL statement in buffer; ask DBMS to compile it */
   sprintf(stmtbuf, "update %s set %s = ? where %s = ?",
                    tblname, searchcol, updatecol);
   exec sql prepare mystmt from :stmtbuf;
   if (sqlca.sqlcode) {
      printf("PREPARE error: %ld\n", sqlca.sqlcode);
      exit();
   }
```

① ②

Zgled:

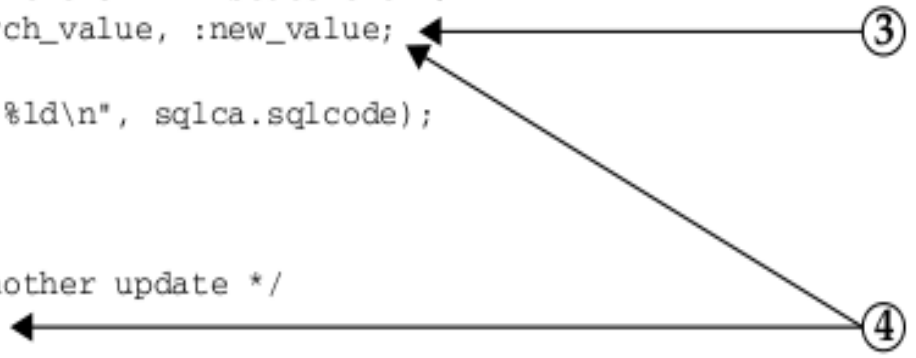
```
   /* Loop prompting user for parameters and performing updates */
   for ( ; ; ) {
      printf("\nEnter search value for %s: ", searchcol);
      scanf("%f", &search_value);
      printf("Enter new value for %s: ", updatecol);
      scanf("%f", &new_value);

      /* Ask the DBMS to execute the UPDATE statement */
      execute mystmt using :search_value, :new_value;
      if (sqlca.sqlcode) {
         printf("EXECUTE error: %ld\n", sqlca.sqlcode);
         exit();
      }

      /* Ask user if there is another update */
      printf("Another (y/n)? ");
      gets(yes_no);
      if (yes_no[0] == 'n')
         break;
   }

   printf("\nUpdates complete.\n");

   exit();
}
```



Zgled: primer izvajanja

```
Enter name of table to be updated:   staff
Enter name of column to be searched: empl_num
Enter name of column to be updated:  quota

Enter search value for empl_num: 106
Enter new value for quota: 150000.00
Another (y/n)? y

Enter search value for empl_num: 102
Enter new value for quota: 225000.00
Another (y/n)? y

Enter search value for empl_num: 107
Enter new value for quota: 215000.00
Another (y/n)? n

Updates complete.
```

Shranjene procedure

- SUPB vedno bolj pogosto vsebujejo zmožnost procesiranja podatkov
  - Proceduralni, objektni, in funkcijski gradniki
  - Shranjeni proceduranі SQL, shranjene procedure
- Visoko-nivojski programski jezik
  - Osnovan na SQL
  - Deklarativno delo s tabelami
  - Vgrajen v jedro SUPB

Shranjene procedure

- Uporaba
  - Običajna vprašanja
  - Jedro aplikacije
  - Sveženjske aplikacije
  - Referenčna integriteta
  - Sistemsko vzdrževanje
  - Vzdrževanje aplikacij

Koncepti shranjenih procedur

- SQL ni bil načrtovan proceduralno
- Programiranje s tabelami
- Koncepti
  - Spremenljivke
  - Pogojno izvajanje
  - Iteracija
  - Bloki
  - Kurzorji
  - Iterator za kurzor
  - Funkcije/Procedure
  - Parametri
  - Prekrivanje
  - Rekurzija
  -

Enostaven primer

- Pridobi številko stranke, ime, bančni limit, ciljni znesek prodaje in prodajalca dodeljenega stranki.
- Dodaj vrstico s podatki o stranki v tabelo strank.
- Popravi kvoto danemu prodajalcu: zvišaj kvoto za specificiran znesek.
- Popravi vrstico pisarne: zvišaj ciljno prodajo za dan znesek.
- Potrdi spremembe, če so vsi prejšnji stavki uspešno izvedeni.

Enostaven primer

- Sekvenca SQL stavkov:

```
INSERT INTO CUSTOMERS (CUST_NUM, COMPANY, CUST_REP, CREDIT_LIMIT)
       VALUES (2137, 'XYZ Corporation', 103, 30000.00);

UPDATE SALESREPS
   SET QUOTA = QUOTA + 50000.00
 WHERE EMPL_NUM = 103;

UPDATE OFFICES
   SET TARGET = TARGET + 50000.00
 WHERE CITY = 'Chicago';

COMMIT;
```

Enostaven primer

- Oracle

```
/* Add a customer procedure */
create procedure add_cust (
   c_name   in varchar2,          /* input customer name */
   c_num    in integer,           /* input customer number */
   cred_lim in number,            /* input credit limit */
   tgt_sls  in number,            /* input target sales */
   c_rep    in integer,           /* input salesrep emp # */
   c_offc   in varchar2)          /* input office city */
as
begin
   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (c_num, c_name, c_rep, cred_lim);

   /* Update row of SALESREPS table */
   update salesreps
      set quota = quota + tgt_sls
    where empl_num = c_rep;

   /* Update row of OFFICES table */
   update offices
      set target = target + tgt_sls
    where city = c_offc;

   /* Commit transaction and we are done */
   commit;
end;
```

Kreiranje shranjene procedure

- Stavek `CREATE PROCEDURE`
  - Ime shranjene procedure
  - Število/tipi parametrov
  - Imena in tipi lokalnih spremenljivk
  - Sekvenca stavkov
- Stavek `DROP PROCEDURE`
- Glej primer `add_cust`
  - Vsi parametri so vrste `IN`
  - Imamo tudi parametre vrste `OUT` in `IN/OUT`

Enostaven primer

- Sybase/Informix

```
/* Add a customer procedure */
create proc add_cust
   @c_name   varchar(20),              /* input customer name */
   @c_num    integer,                  /* input customer number */
   @cred_lim decimal(9,2),             /* input credit limit */
   @tgt_sls  decimal(9,2),             /* input target sales */
   @c_rep    integer,                  /* input salesrep emp # */
   @c_offc   varchar(15)               /* input office city */
as
begin
   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (@c_num, @c_name, @c_rep, @cred_lim)

   /* Update row of SALESREPS table */
   update salesreps
      set quota = quota + quota + @tgt_sls
    where empl_num = @c_rep

   /* Update row of OFFICES table */
   update offices
      set target = target + @tgt_sls
    where city = @c_offc

   /* Commit transaction and we are done */
   commit trans
end
```

```
/* Add a customer procedure */
create procedure add_cust (
   c_name   varchar(20),               /* input customer name */
   c_num    integer,                   /* input customer number */
   cred_lim numeric(16,2),             /* input credit limit */
   tgt_sls  numeric(16,2),             /* input target sales */
   c_rep    integer,                   /* input salesrep emp # */
   c_offc   varchar(15))               /* input office city */

   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (c_num, c_name, c_rep, cred_lim);

   /* Update row of SALESREPS table */
   update salesreps
      set quota = quota + quota + tgt_sls
    where empl_num = c_rep;

   /* Update row of OFFICES table */
   update offices
      set target = target + tgt_sls
    where city = c_offc;

   /* Commit transaction and we are done */
   commit work;
end procedure;
```

Klicanje shranjene procedure

- Vrste klicov
  - 1) Iz aplikacijskega programa, 2) iz druge rutine, in 3) iz interaktivnega okolja
- Klic add_cust iz PL/SQL
  - 1) Iz interaktivnega okolja, 2) iz programa in 3) z poimenovanjem parametrov

PL/SQL

```
EXECUTE ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103,
                 'Chicago');

ADD_CUST('XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago');
```

Transact-SQL

```
EXECUTE ADD_CUST (c_name = 'XYZ Corporation',
                  c_num = 2137,
                  cred_lim = 30000.00,
                  c_offc = 'Chicago',
                  c_rep = 103,
                  tgt_sales = 50000.00);
```

Spremenljivke shranjenih procedur

- Spremenljivke se običajno definirajo za paremetri in pred kodo
- Tipi spremenljivk so lahko vsi SQL tipi

Spremenljivke shranjenih procedur

PL/SQL

```
/* Check order total for a customer */
create procedure chk_tot (c_num in number)
as
   /* Declare two local variables */
   tot_ord  number(16,2);
   msg_text varchar(30);

begin
   /* Calculate total orders for requested customer */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* Load appropriate message, based on total */
   if tot_ord < 30000.00 then
      msg_text := 'high order total';
   else
      msg_text := 'low order total';
   end if;

   /* Do other processing for message text */
   . . .

end;
```

Spremenljivke shranjenih procedur

- Sybase/Informix

```
/* Check order total for a customer */
create proc chk_tot
   @c_num integer      /* one input parameter */
as

   /* Declare two local variables */
   declare @tot_ord money, @msg_text varchar(30)

   begin
      /* Calculate total orders for customer */
      select @tot_ord = sum(amount)
        from orders
       where cust = @c_num

      /* Load appropriate message, based on total */
      if tot_ord < 30000.00
         select @msg_text = "high order total"
      else
         select @msg_text = "low order total"

      /* Do other processing for message text */
      . . .

   end
```

```
/* Check order total for a customer */
create procedure chk_tot (c_num integer)

   /* Declare two local variables */
   define tot_ord  numeric(16,2);
   define msg_text varchar(30);

   /* Calculate total orders for requested customer */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* Load appropriate message, based on total */
   if tot_ord < 30000.00
      let msg_text = "high order total"
   else
      let msg_text = "low order total"

   /* Do other processing for message text */
   . . .

end procedure;
```

Bloki

- Stavke grupiramo v bloke
  - Blok se obnaša kot en stavek
  - Na začetku bloka se definirajo spremenljivke
  - Izjeme so vezane na blok

Bloki

- Različni dialekti SQL/PSM
  - Različna sintaksa definicije spremenljivk

```
/* Informix SPL block of statements */
/* Declaration of any local variables */
define . . .

/* Declare handling for exceptions */
on exception . . .

/* Define the sequence of SQL statements */
begin . . .

end
```

```
/* Oracle PL/SQL statement block */
/* Declaration of any local variables */
declare . . .

/* Specify the sequence of statements */
begin . . .

/* Declare handling for exceptions */
exception . . .

end;
```

```
/* Transact-SQL block of statements */
begin
   /* Sequence of SQL statements appears here */
   . . .
end
```

Funkcije

- Funkcija vrne en objekt
  - Vrednost, objekt, XML dokument
  - Uporaba stavka RETURN; tip je tudi definiran
- Uporaba funkcij
  - Za definicijo stolpcev SELECT stavka
  - Za definicijo pogoja SELECT stavka

```
SELECT COMPANY, NAME
  FROM CUSTOMERS, SALESREPS
 WHERE CUST_REP = EMPL_NUM
   AND GET_TOT_ORDS(CUST_NUM) > 10000.00;
```

Funkcije

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in number)
                return number
as

/* Declare one local variable to hold the total */
tot_ord number(16,2);

begin
   /* Simple single-row query to get total */
   select sum(amount) into tot_ord
     from orders
    where cust = c_num;

   /* return the retrieved value as fcn value */
   return tot_ord;
end;
```

Parametri

- Prenos parametrov po vrednosti in po referenci
- Načini (mode) delovanja parametrov:
  - IN: konstanta v podprogramu
  - OUT: privzeta vrednost
  - IN OUT: dejanski parameter se prepiše v formalnega, na koncu se rezultat prepiše v dejanskega
  - NOCOPY: lahko se prenaša referenca

Primer

```
CREATE OR REPLACE PROCEDURE print (x PLS_INTEGER) IS
BEGIN
  IF x IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE(x);
  ELSE
    DBMS_OUTPUT.PUT_LINE('NULL');
  END IF;
END print;
/
CREATE OR REPLACE PROCEDURE p (
  a        PLS_INTEGER,  -- IN by default
  b     IN PLS_INTEGER,
  c    OUT PLS_INTEGER,
  d IN OUT BINARY_FLOAT
) IS
BEGIN
  -- Print values of parameters:

  DBMS_OUTPUT.PUT_LINE('Inside procedure p:');
  DBMS_OUTPUT.PUT('IN a = '); print(a);
  DBMS_OUTPUT.PUT('IN b = '); print(b);
  DBMS_OUTPUT.PUT('OUT c = '); print(c);
  DBMS_OUTPUT.PUT_LINE('IN OUT d = ' || TO_CHAR(d));

  -- Can reference IN parameters a and b,
  -- but cannot assign values to them.

  c := a+10;  -- Assign value to OUT parameter
  d := 10/b;  -- Assign value to IN OUT parameter
END;
/
```

```
DECLARE
  aa  CONSTANT PLS_INTEGER := 1;
  bb  PLS_INTEGER  := 2;
  cc  PLS_INTEGER  := 3;
  dd  BINARY_FLOAT := 4;
  ee  PLS_INTEGER;
  ff  BINARY_FLOAT := 5;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Before invoking procedure p:');
  DBMS_OUTPUT.PUT('aa = '); print(aa);
  DBMS_OUTPUT.PUT('bb = '); print(bb);
  DBMS_OUTPUT.PUT('cc = '); print(cc);
  DBMS_OUTPUT.PUT_LINE('dd = ' || TO_CHAR(dd));

  p (aa, -- constant
     bb, -- initialized variable
     cc, -- initialized variable
     dd  -- initialized variable
  );

  DBMS_OUTPUT.PUT_LINE('After invoking procedure p:');
  DBMS_OUTPUT.PUT('aa = '); print(aa);
  DBMS_OUTPUT.PUT('bb = '); print(bb);
  DBMS_OUTPUT.PUT('cc = '); print(cc);
  DBMS_OUTPUT.PUT_LINE('dd = ' || TO_CHAR(dd));

  DBMS_OUTPUT.PUT_LINE('Before invoking procedure p:');
  DBMS_OUTPUT.PUT('ee = '); print(ee);
  DBMS_OUTPUT.PUT_LINE('ff = ' || TO_CHAR(ff));

  p (1,        -- literal
     (bb+3)*4, -- expression
     ee,       -- uninitialized variable
     ff        -- initialized variable
   );

  DBMS_OUTPUT.PUT_LINE('After invoking procedure p:');
  DBMS_OUTPUT.PUT('ee = '); print(ee);
  DBMS_OUTPUT.PUT_LINE('ff = ' || TO_CHAR(ff));
END;
/
```

Primer

Result:

```
Before invoking procedure p:
aa = 1
bb = 2
cc = 3
dd = 4.0E+000
Inside procedure p:
IN a = 1
IN b = 2
OUT c = NULL
IN OUT d = 4.0E+000
After invoking procedure p:
aa = 1
bb = 2
cc = 11
dd = 5.0E+000
Before invoking procedure p:
ee = NULL
ff = 5.0E+000
Inside procedure p:
IN a = 1
IN b = 20
OUT c = NULL
IN OUT d = 5.0E+000
After invoking procedure p:
ee = 11
ff = 5.0E-001
```

Prekrite funkcije

- Funkcije in procedure so lahko prekrite (angl. overloaded)
  - Imajo isto ime in različne parametre oz. različne tipe parametrov
- Povezovanje med imenom in kodo procedur in funkcij je definirano na osnovi signatur
- Razlikovanje v NUMERIC tipih
  - BINARY_INTEGER, BINARY_FLOAT, NUMERIC, ...
- Nekateri tipi parametrov ne kreirajo različne signature!
  - Parametri se razlikujejo samo v podtipih (isti nadtip)

Prekrite funckije

```
DECLARE
  TYPE date_tab_typ IS TABLE OF DATE   INDEX BY PLS_INTEGER;
  TYPE num_tab_typ  IS TABLE OF NUMBER INDEX BY PLS_INTEGER;

  hiredate_tab  date_tab_typ;
  sal_tab       num_tab_typ;

  PROCEDURE initialize (tab OUT date_tab_typ, n INTEGER) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Invoked first version');
    FOR i IN 1..n LOOP
      tab(i) := SYSDATE;
    END LOOP;
  END initialize;

  PROCEDURE initialize (tab OUT num_tab_typ, n INTEGER) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Invoked second version');
    FOR i IN 1..n LOOP
      tab(i) := 0.0;
    END LOOP;
  END initialize;

BEGIN
  initialize(hiredate_tab, 50);
  initialize(sal_tab, 100);
END;
/
```

Vračanje vrednosti preko parametrov

- Uporaba OUT parametrov
- Spremenljivke morajo biti pripravljene

```
/* Get customer name, salesrep, and office */
create procedure get_cust_info(c_num  in  number,
                               c_name out varchar,
                               r_name out varchar,
                               c_offc out varchar)
as
begin
   /* Simple single-row query to get info */
   select company, name, city
     into c_name, r_name, c_offc
     from customers, salesreps, offices
    where cust_num = c_num
      and empl_num = cust_rep
      and office = rep_office;
end;
```

```
/* Get the customer info for customer 2111 */
declare the_name varchar(20);
        the_rep  varchar(15);
        the_city varchar(15);
execute get_cust_info(2111, the_name, the_rep, the_city);
```

Stavek RETURN

- Zaključitev izvajanja procedure, funkcije, anonimnega bloka
  - PROCEDURE --- vrne kontrolo (C break())
  - FUNCTION --- vrne kontrolo in vrednost
    - Vsaka pot mora voditi do RETURN !
  - Anonimni blok:

```
BEGIN
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Inside inner block.');
    RETURN;
    DBMS_OUTPUT.PUT_LINE('Unreachable statement.');
  END;
  DBMS_OUTPUT.PUT_LINE('Inside outer block. Unreachable statement.');
END;
/
```

Rekurzija

```
CREATE OR REPLACE FUNCTION fibonacci (
  n PLS_INTEGER
) RETURN PLS_INTEGER
IS
  fib_1 PLS_INTEGER := 0;
  fib_2 PLS_INTEGER := 1;
BEGIN
  IF n = 1 THEN                                  -- terminating condition
    RETURN fib_1;
  ELSIF n = 2 THEN
    RETURN fib_2;                               -- terminating condition
  ELSE
    RETURN fibonacci(n-2) + fibonacci(n-1);  -- recursive invocations
  END IF;
END;
/
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT(fibonacci(i));
    IF i < 10 THEN
      DBMS_OUTPUT.PUT(', ');
    END IF;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE(' ...');
END;
/
```

Pogojno izvajanje

- Stavek IF-THEN-ELSE
- Poglejmo proceduro `add_cust`
  - Dodatna logika za določanje kvote prodajalca

Pogojno izvajanje

```
/* Add a customer procedure */
create procedure add_cust (
   c_name     in varchar2,    /* input customer name */
   c_num      in number,      /* input customer number */
   cred_lim   in number,      /* input credit limit */
   tgt_sls    in number,      /* input target sales */
   c_rep      in number,      /* input salesrep empl # */
   c_offc     in varchar2)    /* input office city */
as
begin
   /* Insert new row of CUSTOMERS table */
   insert into customers (cust_num, company, cust_rep, credit_limit)
          values (c_num, c_name, c_rep, cred_lim);

   if tgt_sales <= 20000.00
   then
      /* Update row of SALESREPS table */
      update salesreps
         set quota = quota + quota + tgt_sls
       where empl_num = c_rep;
   else
      /* Update row of SALESREPS table */
      update salesreps
         set quota = quota + quota + 20000.00
       where empl_num = c_rep;
   end if;

   /* Update row of OFFICES table */
   update offices
      set target = target + tgt_sls
    where city = c_offc;

    /* Commit transaction and we are done */
   commit;
end;
```

Iteracija

- Zanke: `FOR`, `LOOP`
- Izstop iz zanke: `EXIT`

```
/* Process each of ten items */
for item_num in 1..10 loop
   /* Process this particular item */
   . . .

   /* Test whether to end the loop early */
   exit when (item_num = special_item);
end loop;
```

```
/* Repeatedly process some data */
loop
   /* Do some kind of processing each time */
   . . .

   /* Test whether to end the loop early */
   exit when (test_value = exit_value);
end loop;
```

Iteracija

- Zanka `WHILE`

```
/* Lower targets until total below $2,400,000 */
select sum(target) into total_tgt from offices;
while (total_tgt < 2400000.00)
loop
   update offices
      set target = target - 10000.00;
   select sum(target) into total_tgt from offices;
end loop;
```

- Shranjene pocedure imajo bogat nabor gradnikov za kontrolo iteracije
  - Izstop iz zanke: variante `exit`
  - Nadaljevanje zanke: variante `continue`

Iteracija na osnovi kurzorja

- Pregled rezultatov poizvedbe
  - Vrstico za vrstico
  - Alternativa vgnezdenem SQL
- Kursor je gradnik shranjenih procedur
  - Ni potrebno prenašati vrednosti v gostiteljski jezik
  - Definirana je zanka `for` na osnovi kurzorja
- Zgled uporabe kurzorja v PL/SQL
  - Razdelitev naročil v velika in mala naročila

Iteracija na osnovi kurzorja

```
create procedure sort_orders()
   /* Cursor for the query */
   cursor o_cursor is
   select amount, company, name
     from orders, customers, salesreps
    where cust = cust_num
      and rep = empl_num;

   /* Row variable to receive query results values */
   curs_row o_cursor%rowtype;

begin

   /* Loop through each row of query results */
   for curs_row in o_cursor
   loop

      /* Check for small orders and handle */
      if (curs_row.amount < 1000.00)
      then insert into smallorders
                values (curs_row.name, curs_row.amount);

      /* Check for big orders and handle */
      elsif (curs_row.amount > 10000.00)
      then insert into bigorders
                values (curs_row.company, curs_row.amount);
      end if;
   end loop;
  commit;
  end;
```

Preostali gradniki

- Delo z napakami
  - Uporaba izjem
- Zunanje shranjene procedure
  - Povezava z vrsto programskih jezikov
  - Uporaba vgnezdenega SQL

Prednosti shranjenih procedur

- Hitrost izvajanja programov
  - Koda je prevedena in shranjena v PB
  - Jedro SUPB izvaja kodo
- Ponovna uporabnost
  - Kodo lahko uporabljamo na različnih mestih
- Zmanjšan promet po omrežju
  - Ni potrebno prenašati podatkov do aplikacije

Prednosti shranjenih procedur

- Varnost
  - Procedura je objekt shranjen v PB
  - Privilegije imamo lahko samo do procedure ne tabel
- Enkapsulacija
  - Shranjene rutine omogočajo zasnovo objektno-usmerjenega sistema
  - Metode razredov
- Enostavnost dostopa
  - Klic rutine

Prednosti shranjenih procedur

- Zagotavljanje poslovnih pravil
  - Koda se lahko definira na enem mestu za vse uporabnike (GUI, Web, interaktiven SQL)
  - Poslovno pravilo zapisano v SQL/PSM

Programiranje s tabelami

- Tabela je osnovni objekt programiranja
- Prepisovanje podatkov iz tabele v tabelo
- Filtriranje, sortiranje in rekonstrukcija tabel
- Zanemarimo kompleksnost korakov
- Deklarativno programiranje
- Običajno se v repozitoriju nabere veliko tabel

Sistemske shranjene procedure

- Relacijski sistemi definirajo sistemske procedure na osnovi shranjenih procedur
  - Delo z uporabniki
  - Delo s privilegiji in grupami
  - Delo s porazdeljenimi strežniki
  - Replikacija tabel
  - ...
- Sybase je bil pionir področja

Prožilci

- Niso definirani v SQL2 (92)
  - Ni se pričakovalo, da bodo uspeli med uporabniki
  - Vsi sistemi imajo prožilce že od 1990
- Aktivne podatkovne baze
  - Pravila uporabljena nad podatki v bazi
  - Ekspertni sistemi

Referenčna integriteta

- Pravila, ki veljajo za stolpce, ki referencirajo zapise v drugi tabeli
  - Tuj kjuč mora obstajati v referencirani tabeli
  - Akcije:
    - Kaj če se zbriše referenciran ključ?
    - Kaj če se doda neobstječ ključ?
- Lahko vidimo kot pravila
  - Prožilci se aktivirajo ob spremembah zapisov

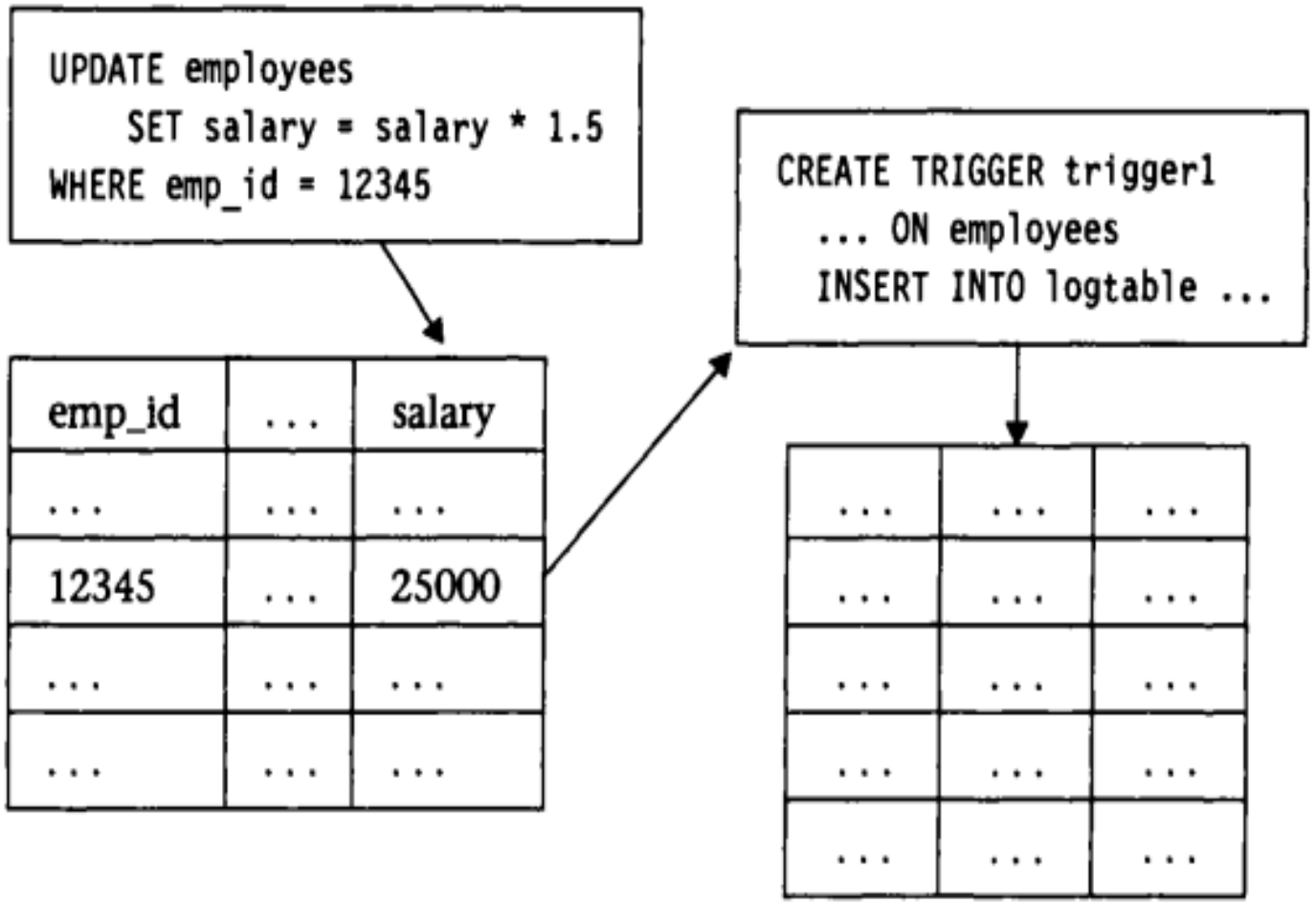
Prožilci

- Prožilci so fleksibilni glede pogoja aktivacije
  - Vstavljanje, popravljanje in brisanje zapisov
- Aktivacija vezana na:
  - INSERT, UPDATE ali DELETE stavek
  - Dodajanje, spreminjanje oz. brisanje posameznih zapisov
- Objekti ob aktivaciji prožilca:
  - Subjektna tabela
  - Aktivacijski SQL stavek
  - Aktiviran SQL stavek

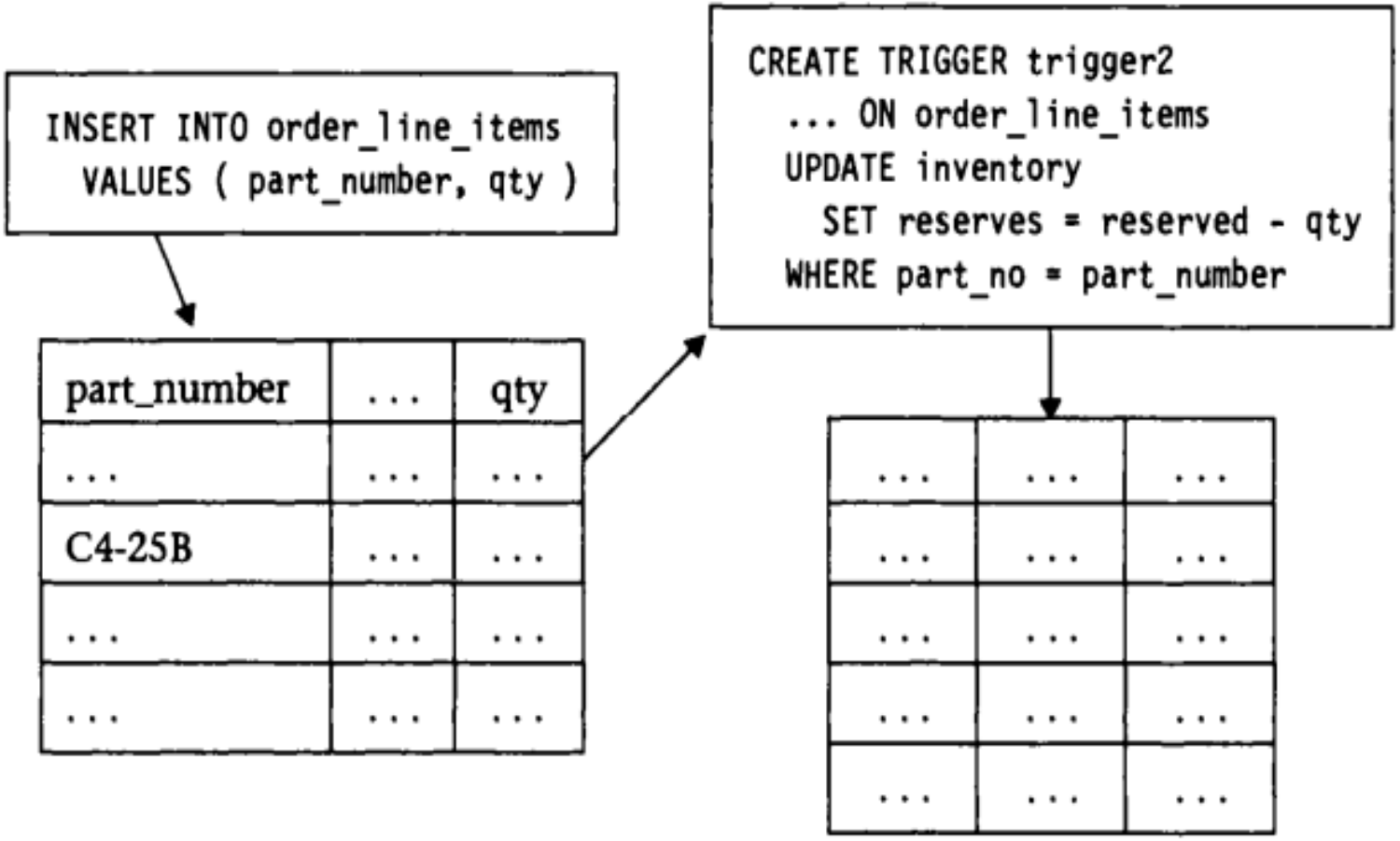
Prožilci

- Prožilci so lahko aktivirani pred ali po aktivacijskem dogodku
- Uporaba prožilcev:
  - Dnevniki akcij
  - Posledice akcij
  - Vzpostavljanje konsistence
  - Aktivacija procedur izven SPUB
- Uporabnik mora imeti privilegije za delo s prožilcem

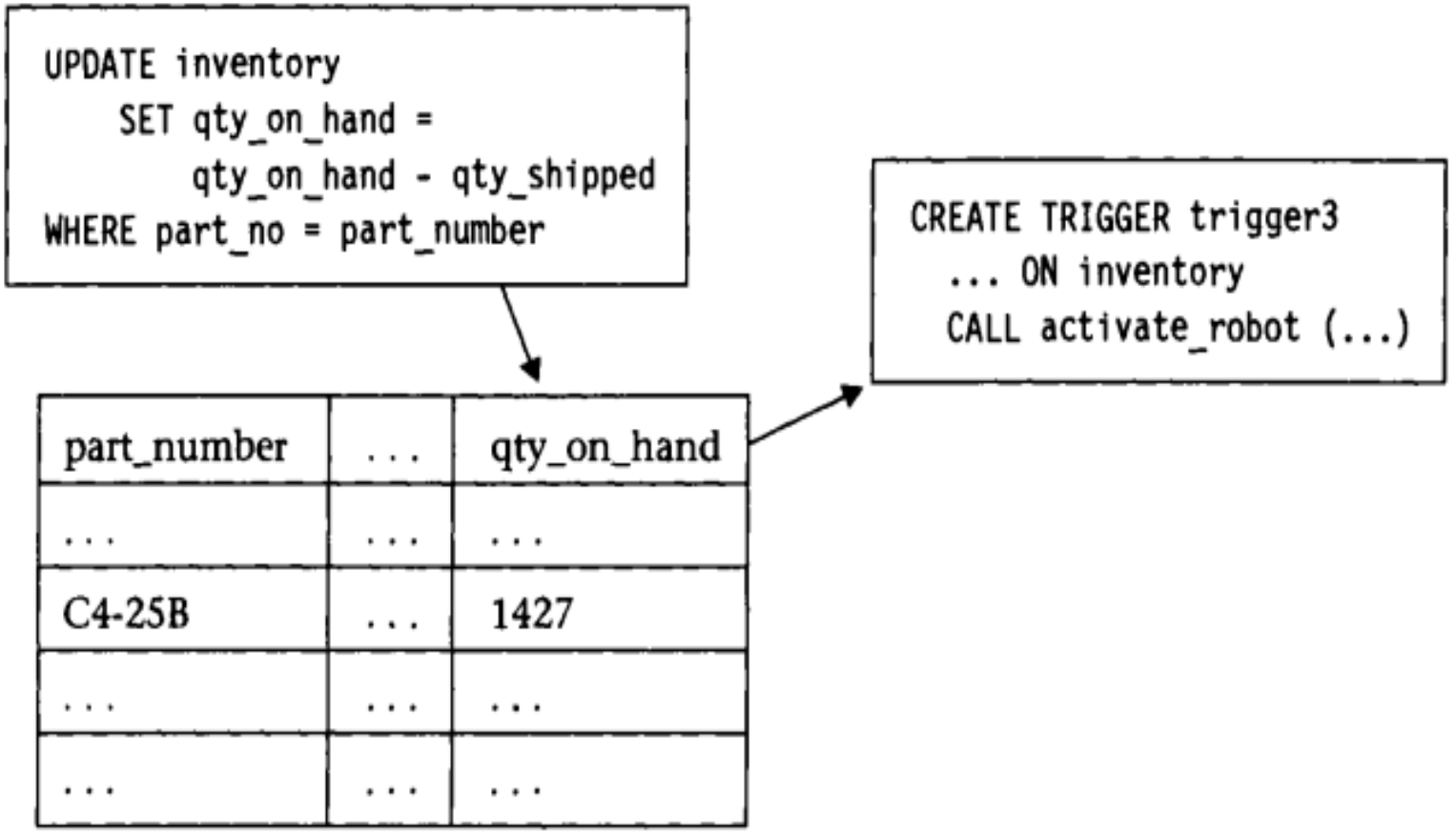
Dnevniki


UPDATE employees
SET salary = salary * 1.5
WHERE emp_id = 12345
CREATE TRIGGER trigger1
... ON employees
INSERT INTO logtable ...
emp_id
...
salary
12345
25000

Vzpostavljanje konsistence



Akcije izven SUPB



Sintaksa prožilcev

- Delitev na `BEFORE`, `AFTER`, `INSTEAD OF`
- Delitev na `INSERT`, `UPDATE`, `DELETE`
  - Vsak razred pokriva več vrst poizvedb
  - Aktivacija s SELECT je možna v nakaterih SUPB
- Prožilci na stavkih in na izbranih zapisih
- Pogoj prožitve
  - Akcija proženja ima lahko pogoj
  - Sistem pravil - Ekspertni sistemi

Osnovni elementi prožilca

MovieExec(name, address, cert#, netWorth)

```
1)  CREATE TRIGGER NetWorthTrigger
2)  AFTER UPDATE OF netWorth ON MovieExec
3)  REFERENCING
4)      OLD ROW AS OldTuple,
5)      NEW ROW AS NewTuple
6)  FOR EACH ROW
7)  WHEN (OldTuple.netWorth > NewTuple.netWorth)
8)      UPDATE MovieExec
9)      SET netWorth = OldTuple.netWorth
10)     WHERE cert# = NewTuple.cert#;
```

Prepreči spremembo vrednosti filma na manjšo vrednost.

Osnovni elementi prožilca

- Vrstica (1): Stavek CREATE TRIGGER
- Vrstica (2): BEFORE | AFTER | INSTEAD OF

  INSERT | UPDATE | DELETE

  OF <atribut> -- fokus na atribut (update)
- Vrstice (4,5): OLD | NEW | PARENT

  ROW | TABLE AS
- Vrstica (6): FOR EACH ROW | FOR EACH STATEMENT

Sintaksa prožilcev

```
<trigger definition> ::=
    CREATE TRIGGER <trigger name>
      <trigger action time> <trigger event>
      ON <table name> [ REFERENCING <old or new values alias list> ]
      <triggered action>

<trigger action time> ::=
    BEFORE
  | AFTER

<trigger event> ::=
    INSERT
  | DELETE
  | UPDATE [ OF <trigger column list> ]

<trigger column list> ::= <column name list>

<triggered action> ::=
    [ FOR EACH { ROW | STATEMENT } ]
      [ WHEN <left paren> <search condition> <right paren> ]
      <triggered SQL statement>
```

Sintaksa prožilcev

```
<triggered SQL statement> ::=
    <SQL procedure statement>
  | BEGIN ATOMIC
      { <SQL procedure statement> <semicolon> }...
    END

<old or new values alias list> ::=
    <old or new values alias>...

<old or new values alias> ::=
    OLD [ROW ][AS ]<old values correlation name>
  | NEW [ROW ][AS ]<new values correlation name>
  | OLD TABLE [ AS ] <old values table alias>
  | NEW TABLE [ AS ] <new values table alias>

<old values table alias> ::= <identifier>

<new values table alias> ::= <identifier>

<old values correlation name> ::= <correlation name>

<new values correlation name> ::= <correlation name>
```

Primer STATEMENT prožilca

MovieExec(name, address, cert#, netWorth)

```
 1)  CREATE TRIGGER AvgNetWorthTrigger
 2)  AFTER UPDATE OF netWorth ON MovieExec
 3)  REFERENCING
 4)      OLD TABLE AS OldStuff,
 5)      NEW TABLE AS NewStuff
 6)  FOR EACH STATEMENT
 7)  WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
 8)  BEGIN
 9)      DELETE FROM MovieExec
10)      WHERE (name, address, cert#, netWorth) IN NewStuff;
11)      INSERT INTO MovieExec
12)          (SELECT * FROM OldStuff);
13)  END;
```

Primer prožilca za NULL vrednost

Movies(title, year, length, genre, studioName, producerC#)

```
1)  CREATE TRIGGER FixYearTrigger
2)  BEFORE INSERT ON Movies
3)  REFERENCING
4)      NEW ROW AS NewRow
5)      NEW TABLE AS NewStuff
6)  FOR EACH ROW
7)  WHEN NewRow.year IS NULL
8)  UPDATE NewStuff SET year = 1915;
```

Popravi n-terice, ki se vstavljajo.

Primer INSTEAD OF prožilca

```
CREATE OR REPLACE VIEW order_info AS
   SELECT c.customer_id, c.cust_last_name, c.cust_first_name,
          o.order_id, o.order_date, o.order_status
   FROM customers c, orders o
   WHERE c.customer_id = o.customer_id;
```

```
CREATE OR REPLACE TRIGGER order_info_insert
   INSTEAD OF INSERT ON order_info
   DECLARE
     duplicate_info EXCEPTION;
     PRAGMA EXCEPTION_INIT (duplicate_info, -00001);
   BEGIN
     INSERT INTO customers
       (customer_id, cust_last_name, cust_first_name)
     VALUES (
     :new.customer_id,
     :new.cust_last_name,
     :new.cust_first_name);
   INSERT INTO orders (order_id, order_date, customer_id)
   VALUES (
     :new.order_id,
     :new.order_date,
     :new.customer_id);
   EXCEPTION
     WHEN duplicate_info THEN
       RAISE_APPLICATION_ERROR (
         num=> -20107,
         msg=> 'Duplicate customer or order ID');
   END order_info_insert;
/
```

Literatura

- Paul Weinberg, James Groff, Andrew Oppel. SQL The Complete Reference 3rd Edition, McGraw-Hill, 2010.
- Jim Melton, Alan R.Simon, SQL:1999 Understanding Relational Language Components, Academic Press, 2002
- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, DATABASE SYSTEMS: The Complete Book, Prentice Hall, 2008.
- Oracle Database PL/SQL Language Reference, 11g Release 2 (11.2), E25519-13, 2014.