Kreiranje procedura

Na primer neka je potrebno dodati novog klijenta, a potom osvežiti informacije o dodeljenom prodavcu i ciljnoj kvoti, kao i osvežiti informacije koje se odnose na kancelariju tako da se ciljna prodaja podigne za odgovarajući iznos, i naravno potvrditi promene završavanjem transakcije.

```
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;
```

Ukoliko bi se napravila odgovarajuća procedura, onda bi bila pozivana sa:

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

Svi parametri procedure su ulazni, što se u nekim dijalektima naznačava pomoću ključne reči IN. Ovi parametri se mogu koristiti svuda u telu procedure gde je inače moguće da se pojavi konstanta. Moguća je upotreba i takozvanih izlaznih parametara (ključna reč OUTPUT, ili samo OUT) kada se očekuje da procedura vrati neku vrednost. Takođe, kod nekih implementacija postoje i ulazno-izlazni parametri (INOUT).

Implementacija u ORACLE (PL/SQL)

```
/* 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;



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

Ukoliko se poziva iz druge procedure onda može i samo `ADD_CUST(…)`, odnosno, bez reči `EXECUTE`.

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

Implementacija u SQL Server (T-SQL)

Ne koriste se zagrade u deklaraciji procedure, parametri se označavaju specijalnim karakterom (@), a takođ čitava procedura je jedan SQL iskaz, pa ukoliko se želi nešto kompleksnije telo, onda se to navodi kao blok (begin-end) .

```
/* 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


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

EXEC ADD_CUST 'XYZ Corporation', 2137, 30000.00, 50000.00, 103, 'Chicago';

EXEC ADD_CUST @C_NAME = 'XYZ Corporation',
               @C_NUM = 2137,
           @CRED_LIM = 30000.00,
             @C_OFFC = 'Chicago',
              @C_REP = 103,
            @TGT_SLS = 50000.00;
```

Implementacija u Informix

```
/* 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;

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

Ukoliko se poziva iz druge procedure onda može i sa:

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

Upotreba promenljivih

Svi dijalekti podržavaju upotrebu lokalnih promenljivih unutar tela procedure. Promenljive se deklarišu odmah nakon zaglavlja, a pre ostalih iskaza.

Na primer, deo procedure koji treba da izračuna ukupan iznos neplaćenih narudžbina za određenog klijenta, i postavljanje jedne od dve poruka u zavisnosti da li je taj iznos ispod ili iznad 30000.

Implementacija u Oracle (PL/SQL)

Nazivi promenljivih su klasični SQL identifikatori, a dodela se obavlja kroz SELECT…INTO naredbu, odnosno koristeći format sličan Pascal programskom jeziku (:=).

```
/* 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;
```

Implementacija u SQL Server (T-SQL)

Upotrebljava se DECLARE za deklaraciju. Naziv počinje specijalnim karakterom @, a dodela se obavlja kroz SELECT naredbu, odnosno (SELECT @promenljiva = …).

```
/* 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
```

Implementacija u Informix

Upotrebljava se DEFINE za deklaraciju (ovaj primer pokazuje samo deo opcija koje postoje). Nazivi promenljivih su kao klasični SQL identifikatori. Dodela se obavlja kroz SELECT…INTO naredbu, ili koristeći LET iskaz.

```
/* 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;
```

Blokovi iskaza

Svi dijalekti podržavaju grupisanje sekvence iskaza u okviru jednog bloka.

Implementacija u Oracle (PL/SQL)

Sve tri sekcije su opcione.

```
/* Oracle PL/SQL statement block */
/* Declaration of any local variables */
declare . . .
/* Specify the sequence of statements */
begin . . .
/* Declare handling for exceptions */
exception . . .
end;
```

Implementacija u SQL Server (T-SQL)

Sekcija ima za cilj samo da grupiše iskaze, a ne utiče na doseg i vidljivost lokalnih promenljivih.

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

Implementacija u Informix

Blok iskaza u ovoj implementaciji ne sadrži samo sekvencu iskaza, već opcionu deklaraciju promenljivih vidljivih samo unutar bloka, kao i sekciju u kojoj se razrešavaju greške i izuzeci. Generalno kod Informix blokovi nisu tako često potrebni kao kod T-SQL zbog toga što su kontrolne strukture sa eksplicitnim krajem (IF…END IF, WHILE…END WHILE, FOR…END FOR), baš kao i kod PL/SQL.

```
/* Informix block of statements */
/* Declaration of any local variables */
define . . .
/* Declare handling for exceptions */
on exception . . .
/* Define the sequence of SQL statements */
begin . . .
end
```

Petlje

Svi dijalekti podržavaju razne vrste ponavljanja u izvršavanju.

Implementacija u Oracle (PL/SQL)

```
/* 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;
```

Implementacija u SQL Server (T-SQL)

SQL Server nema FOR petlje, ali se isto može postići upotrebom WHILE petlji.

```
/* Process each of ten items */
while @item_num <11;
begin
set @item_num = @item_num + 1;

/* Process this particular item */
. . .
/* Test whether to end the loop early */
if (@item_num = special_item)
      break
else
      continue
end
```

Implementacija u Informix

```
/* Process each of ten items */
for item_num = 1 to 10 step 1

/* Process this particular item */
. . .
/* Test whether to end the loop early */
if (item_num = special_item)
then exit for;

end for;
```

Oracle i Informix, pored FOR poseduju i WHILE petlje.

Ponavljanje koristeći kursore

Svi dijalekti podržavaju upotrebu kursora u slučajevima kada je potrebno obraditi rezultate nekog upita. Kod svih je konceptualno isti niz koraka: deklaracija, otvaranje, dohvatanje, zatvaranje (DECLARE CURSOR, OPEN CURSOR, FETCH, CLOSE CURSOR).

Implementacija u Oracle (PL/SQL)

Za dohvatanje rezultata upita iz kursora koristiti se specijalna promenljiva koja je tipa CURS_ROW koja odgovara jednom redu kreiranog kursora.

```
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;
```

Implementacija u SQL Server (T-SQL)

SQL Server nema specijalizovanu FOR petlju za rad sa kursorima, već se kontrola i ponavljanje izvršava proveravanjem sistemske promenljive @@SQLSTATUS, koja je ekvivalent standardnom SQLSTATE kodu.

```
create proc sort_orders()
as
/* Local variables to hold query results */
declare @ord_amt decimal(16,2);    /* order amount */
declare @c_name varchar(20);       /* customer name */
declare @r_name varchar(15);       /* salesrep name */

/* Declare cursor for the query */
declare o_curs cursor for
select amount, company, name
from orders, customers, salesreps
where cust = cust_num
and rep = empl_num

begin
/* Open cursor and fetch first row of results */
open o_curs
fetch o_curs into @ord_amt, @c_name, @r_name

/* If no rows, return immediately */
if (@@sqlstatus = 2)
begin
      close o_curs
      return
end

/* Loop through each row of query results */
while (@@sqlstatus = 0)
begin
      /* Check for small orders and handle */
      if (@ord_amt < 1000.00)
            insert into smallorders
                  values (@r_name, @ord_amt)
      /* Check for big orders and handle */
      else if (curs_row.amount > 10000.00)
            insert into bigorders
                  values (@c_name, @ord_amt)
end

/* Done with results; close cursor and return */
close o_curs

end
```

Implementacija u Informix

```
create procedure sort_orders()

/* Local variables to hold query results */
define ord_amt numeric(16,2);      /* order amount */
define c_name varchar(20);         /* customer name */
define r_name varchar(15);         /* salesrep name */

/* Execute query and process each results row */
foreach select amount, company, name
          into ord_amt, c_name, r_name
          from orders, customers, salesreps
         where cust = cust_num and rep = empl_num;
begin

      /* Check for small orders and handle */
      if (ord_amt < 1000.00)
      then insert into smallorders
                  values (r_name, ord_amt);
      /* Check for big orders and handle */
      elif (ord_amt > 10000.00)
      then insert into bigorders
                  values (c_name, ord_amt);
      end if;
end;
end foreach;
end procedure;
```

Vraćanje vrednosti preko parametara

Pored uskladištenih procedura, većina dijalekata podržava i uskladištene korisničke funkcije. Funkcije uvek vraćaju jednu stvar (vrednost, objekat, XML dokument) dok procedura ili ne vraća ništa ili vraća veći broj stvari. Funkcije se obično koriste unutar iskaza za kolone unutar SELECT klauzule upita, tako da bivaju pozvane po jednom za svaki red koji se obrađuje. Na primer, funkcija GET_TOT_ORDS bi mogla biti pozvana:

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

Implementacija u Oracle (PL/SQL)

```
/* 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;
```

Implementacija u SQL Server (T-SQL)
SQL Server takođe ima podršku za funkcije, slično kao kod Oracle i Informix.

Implementacija u Informix

Umesto uvođenja izlaznih parametara, Informix proširuje definiciju funkcija time što omogućava veći broj povratnih vrednosti.

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in integer) returning numeric(16,2)
/* Declare one local variable to hold the total */
define tot_ord numeric(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 function;
```

Vraćanje vrednosti preko parametara

Obično se u slučaju potrebe da se vrati jedna vrednost, koriste uskladištene korisničke funkcije. Međutim, u slučaju kada je potrebno vratiti više vrednosti, većina dijalekata predviđa izlazne parametre.

Implementacija u Oracle (PL/SQL)

```
/* 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;
```

Primer jednog anonimnog bloka u kome se poziva ova procedura je:

```
/* 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);
```

Implementacija u SQL Server (T-SQL)

```
/* Get customer name, salesrep, and office */
create procedure get_cust_info(@c_num integer,
                               @c_name varchar(20) out,
                               @r_name varchar(15) out,
                               @c_offc varchar(15) out)
as
begin

/* Simple single-row query to get info */
select @c_name = company,
@c_offc = city
from customers, salesreps, offices
where cust_num = @c_num
and empl_num = cust_rep
and office = rep_office;
end
```

Prilikom poziva ovakve procedure, mora se naglasiti da koji su parametri izlazni:

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

exec get_cust_info @c_num = 2111,
                   @c_name = the_name output,
                   @r_name = the_rep output,
                   @c_offc = the_city output;
```

Implementacija u Informix

Umesto uvođenja izlaznih parametara, Informix proširuje definiciju funkcija time što omogućava veći broj povratnih vrednosti.

```
/* Get customer name, salesrep, and office */
create function get_cust_info(c_num integer)
      returning varchar(20), varchar(15), varchar(15)

define c_name varchar(20);
define r_name varchar(15);
define r_name varchar(15);

/* Simple single-row query to get info */
select company, name, city
into cname, r_name, c_offc
from customers, salesreps, offices
where cust_num = c_num
and empl_num = cust_rep
and office = rep_office;

/* Return the three values */
return cname, r_name, c_offc;

end procedure;
```

Prilikom poziva ovakve funkcije, mora se koristiti specijalna RETURNING klauzula.

```
/* Get the customer info for customer 2111 */
define the_name varchar(20);
define the_rep varchar(15);
define the_city varchar(15);

call get_cust_info (2111) returning the_name, the_rep, the_city;
```

Obrada grešaka i izuzetaka

Svi dijalekti podržavaju neki način obrade grešaka i izuzetaka.

Implementacija u Oracle (PL/SQL)

```
/* 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 */
declare 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;

exception
/* Handle the situation where no orders found */
when no_data_found
then raise_application_error (-20123, 'Bad cust#');

/* Handle any other exceptions */
when others
then raise_application_error (-20199,'Unknown error');
end;
```

Implementacija u SQL Server (T-SQL)

SQL Server omogućava obradu grešaka upotrebom skupa globalnih promenljivih (njih par od preko 100 drugih globalnih promenljivih koje daju uvid u stanje čitavog servera, transakcija, otvorenih konekcija itd.). Najvažnije promenljive za obradu grešaka su @@ERROR koja sadrži status poslednje izvršenog niza iskaza, kao i @@SQLSTATUS koja sadrži status poslednje FETCH operacije.

Implementacija u Informix

```
/* Return total order amount for a customer */
create function get_tot_ords(c_num in integer)
returning numeric(16,2)

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

/* Define exception handler for error #-123 and -121 */
on exception in (-121, -123)
/* Do whatever is appropriate here */
. . .
end exception;

on exception
/* Handle any other exceptions in here */
. . .
end exception;
```

Okidači

Svi dijalekti podržavaju definiciju okidača, ali se sintaksa i mogućnosti znatno razlikuju.

Implementacija u Oracle (PL/SQL)

Oracle ima podršku za najraznovrsnije opcije okidača. Podržava BEFORE i AFTER okidače, kao i INSTEAD OF okidače, a sve to na nivou iskaza i na nivou reda. Podžava čak i okidače koji reaguju na događaje na nivou sistema (npr. Kada se neki korisnik poveže na bazu, ili kada se izda neka određena komanda nad bazom itd.)

```
create trigger bef_upd_ord
before update on orders
begin
/* Calculate order total before changes */
old_total = add_orders();
end;

create trigger aft_upd_ord
after update on orders
begin
/* Calculate order total after changes */
new_total = add_orders();
end;

create trigger dur_upd_ord
before update of amount on orders
referencing old as pre new as post
/* Capture order increases and decreases */
for each row
when (:post.amount != :pre.amount)
begin
      if post.amount != :pre.amount)
      then
            if (:post.amount < :pre.amount)
            then
                  /* Write decrease data into table */
                  insert into ord_less
                  values (:pre.cust, :pre.order_date, :pre.amount, :post.amount);
            elsif (:post.amount > :pre.amount)
            then
                  /* Write increase data into table */
                  insert into ord_more
                  values (:pre.cust, :pre.order_date, :pre.amount, :post.amount);
            end if;
      end if;
end;
```

```
Create or replace trigger upd_tgt
/* Insert trigger for SALESREPS */
before insert on salesreps
for each row
begin
      if :new.quota is not null
      then
            update offices
            set target = target + new.quota;
      end if;
end;
```

Implementacija u SQL Server (T-SQL)

SQL Server pruža podršku za AFTER i INSTEAD OF okidače, i to samo na nivou iskaza, ali se upotrebom globalne promenljive može proveriti da li je iskaz promenio jedan ili više redova.

```
create trigger upd_tgt
/* Insert trigger for SALESREPS */
on salesreps
for insert
as
if (@@rowcount = 1)
      begin
            update offices
            set target = target + inserted.quota
            from offices, inserted
            where offices.office = inserted.rep_office;
      end
else
      raiserror 23456;
```

```
create trigger chk_del_cust
/* Delete trigger for CUSTOMERS */
on customers
for delete
as
/* Detect any orders for deleted cust #'s */
if (select count(*)
    from orders, deleted
    where orders.cust = deleted.cust_num) > 0
begin
      rollback transaction
      print "Cannot delete; still have orders"
      raiserror 31234
end;
```

```
create trigger upd_reps
/* Update trigger for SALESREPS */
on salesreps
for insert, update
if update(quota)
/* Handle updates to quota column */
. . .
if update (sales)
/* Handle updates to sales column */
. . .
```

Implementacija u Informix

Informix pruža podršku za BEFORE i AFTER okidače, ali pored podrške za okidače na nivou iskaza, ima podršku i za okidače na nivou reda.

```
create trigger new_sls
insert on salesreps . . .

create trigger del_cus_chk
delete on customers . . .

create trigger ord_upd
update on orders . . .

create trigger sls_upd
update of quota, sales on salesreps . . .


create trigger upd_ord
update of amount on orders
referencing old as pre new as post

/* Calculate order total before changes */
before (execute procedure add_orders() into old_total;)

/* Capture order increases and decreases */
for each row
when (post.amount < pre.amount)
/* Write decrease data into table */
(insert into ord_less
values (pre.cust, pre.order_date, pre.amount, post.amount);)
when (post.amount > pre.amount)
/* Write increase data into table */
(insert into ord_more
values (pre.cust, pre.order_date, pre.amount, post.amount);)
/* After changes, recalculate total */
after (execute procedure add_orders() into new_total;)
```