Oracle PL/SQL

widoki, procedury, funkcje, triggery

Oracle PL/SQL

- Koncepcja języka PL/SQL
- Blok
- Zmienne
- Podstawowe konstrukcje sterujące
- Widoki
- Procedury
- Funkcje
- Triggery

Blok anonimowy

```
[DECLARE]
  -- deklaracje
BEGIN
  -- polecenia
[EXCEPTION]
  -- obsługa błędów, wyjątków
END;
```

Zmienne

- Deklarowana w sekcji DECLARE
- Rodzaje zmiennych:
  - proste
    - liczba
    - ciąg znaków,
    - data
    - wartość logiczna
  - złożone
    - rekord
    - tablica
- Widoczne w bloku deklaracji i blokach zagnieżdżonych.

Zmienne proste

```
declare
    s varchar2(10);
    a int;
    b int;
begin
    s := 'abc';
    a := 1;
    b := a + 7;

    dbms_output.Put_line(s);
    dbms_output.Put_line(b);
end;
```

Rekord - typ i zmienna

```
declare
    type person is record
                    (
                        first varchar2(50),
                        last varchar2(50)
                    );
    o person;
begin
    o.first := 'jan';
    o.last := 'kowalski';

    dbms_output.Put_line(o.last);
end;
```

%TYPE, %ROWTYPE

```oracle
select * from employees;

declare
    f employees.firstname%TYPE;
begin
    f := 'john';
    dbms_output.Put_line(f);

end;
```

```oracle
declare
    f employees%ROWTYPE;
begin
    f.firstname := 'john';
    dbms_output.Put_line('firstname: ' || f.firstname);

end;
```

IF

```
declare
    b boolean := true;
begin
    if b then
        dbms_output.put_line('true');
        dbms_output.put_line('true');
    else
        dbms_output.put_line('false');
    end if;
end;
```

CASE

```
declare
    s varchar2(10) := 'abc';
begin
    case
        when s = 'abc' then
            dbms_output.put_line(1);
        when s = 'def' then
            dbms_output.put_line(2);
        else
            dbms_output.put_line(3);
    end case;
end;
```

LOOP

```
declare
    i int := 0;
begin
    loop
        if i > 5 then
            exit;
        end if;
        dbms_output.put_line(i);
        i := i + 1;
    end loop;
end;
```

WHILE, FOR

```
declare
    i int := 0;
begin
    while i <= 5 loop
        dbms_output.put_line(i);
        i := i +1;
    end loop;
end;
```

```
declare
    i int := 0;
begin
    for  i in 1..5 loop
        dbms_output.put_line(i);
    end loop;
end;
```

Wyjątki

```
declare
    i int;
    exc1 exception;
begin
    i := 3;

    if i = 1 then
        raise exc1;
    end if;
    if i = 3 then
        raise_application_error(-20001, 'exc3');
    end if;
    dbms_output.put_line('OK');

exception
    when exc1 then
        dbms_output.put_line('exc 1');
        raise;
    when others then
        dbms_output.put_line('other exc');

end;
```

Wyjątki c.d.

```
declare
   i int;
   exc1 exception;
   exc2 exception;
   exc3 exception;
   pragma exception_init(exc3,-20003);

begin
   i := 3;

   if i = 1 then
      raise exc1;
   end if;
   if i = 2 then
       raise exc2;
   end if;
   if i = 3 then
       raise_application_error(-20003, 'exc3 message');
   end if;
   dbms_output.put_line('OK');

exception
   when exc1 then
      dbms_output.put_line('exc 1');
   when exc3 then
      dbms_output.put_line('exc 3');
      dbms_output.put_line('error code: ' || sqlcode);
      dbms_output.put_line('error message: ' || sqlerrm);
      raise;
   when others then
       dbms_output.put_line('other exc');
end;
```

SELECT ... INTO

```
declare
    f employees%ROWTYPE;
begin
    select * into f from employees where employeeid  = 1;
    dbms_output.Put_line('firstname: ' || f.firstname);

end;
```

```
declare
    first varchar2(50);
    last varchar2(50);
begin
    select firstname, lastname into first, last from employees where employeeid  = 1;
    dbms_output.Put_line('firstname: ' || first || ' lastname: ' || last);

end;
```

INSERT ... RETURNING

```
declare
    companyname varchar(50); phone varchar(50);
    id int;
begin
    insert into shippers(companyname, phone)
    values ('Taxi', '123')
    returning shipperid into id;

    dbms_output.Put_line('id: ' || id);

end;
```

```
declare
  s shippers%rowtype;
begin
  insert into shippers(companyname, phone)
  values ('Taxi2', '123')
  returning shipperid, companyname into s.shipperid, s.companyname;

  sys.dbms_output.Put_line('id: ' || s.shipperid || ' name : ' || s.companyname);

end;
```

UPDATE

```
declare
    s shippers%rowtype;
begin
    s.shipperid := 1;
    s.companyname := 'John';
    s.phone := '123';

    update shippers
    set row = s
    where shipperid = 1 ;
end;
```

Wyjątki c.d

```sql
drop table test1;

create table test1
(
    tid int  generated always as identity not null,
    tname varchar(100),
    status char(1),
    constraint test1_pk primary key ( tid ) enable
);


alter table test1
add constraint test1_chk1 check
(status in ('A','B')) enable;
```

Wyjątki c.d

```oracle
begin
    insert into test1(tname, status)
    values ('ala', 'A');

    insert into test1(tname, status)
    values ('bala', 'X');

    dbms_output.put_line('OK');
end;
```

> Error
> [23000][2290] ORA-02290: check constraint (NORTHWIND_M.TEST1_CHK1) violated
> ORA-06512: at line 5 Position: 0

- żaden wiersz nie zostanie dopisany

Wyjątki c.d

```
begin
    insert into test1(tname, status)
    values ('ala', 'A');

    insert into test1(tname, status)
    values ('bala', 'X');

    dbms_output.put_line('OK');

exception
    when others then
        dbms_output.put_line('other exc');
end;
```

- pierwszy wiersz zostanie dopisany
  - jeśli jesteśmy w trybie `Auto-Commit` to wiersz zostanie dopisany trwale
  - jeśli nie jesteśmy w trybie `Manual` - pozostaniemy "wewnątrz transakcji"
    - można zrobić `rollback`

Wyjątki c.d

```oracle_plsql
begin
    insert into test1(tname, status)
    values ('ala', 'A');

    insert into test1(tname, status)
    values ('bala', 'X');

    dbms_output.put_line('OK');

exception
    when others then
        dbms_output.put_line('other exc');
        raise;
end;
```

> Error
> [23000][2290] ORA-02290: check constraint (NORTHWIND_M.TEST1_CHK1) violated
> ORA-06512: at line 5 Position: 0

- żaden wiersz nie zostanie dopisany

Widoki

- Sposób definiowania widoków jest bardzo podobny w zasadzie we wszystkich SZBD
  - ale trzeba pamiętać o różnicach w składni pomiędzy poszczególnymi dialektami SQL

Widoki

- tworzenie widoku

```
create or replace vw_emp_names
as
select firstname || ' ' || lastname as name
from employees
```

```
select * from vw_emp_names
```

wynik:

	name
1	Andrew Fuller
2	Nancy Davolio
3	Janet Leverling
4	Margaret Peacock

- usunięcie widoku

```
drop view vw_emp_names;
```

Widoki c.d.

```sql
create or alter view vw_products_ok
as
select productid, categoryid, supplierid, productname, unitprice, unitsinstock
from products
where discontinued = 0;
```

```sql
select * from vw_products_ok;
```

wynik (fragment):

	productid	categoryid	supplierid	productname	unitprice	unitsinstock
1	1	1	1	Chai	18	39
2	2	1	1	Chang	19	17
3	3	2	1	Aniseed Syrup	10	13
4	4	2	2	Chef Anton's Cajun Seasoning	22	53

Widoki c.d.

```
create or replace view vw_avail_products
as
select productid, categoryid, supplierid, productname, unitprice, unitsinstock
from vw_products_ok
where unitsinstock > 0;
```

```
select count(*) from vw_products_ok
```

wynik:

	COUNT(*)
1	69

```
select count(*) from vw_avail_products;
```

wynik:

	COUNT(*)
1	68

Widoki c.d.

```sql
create or replace view vw_order_details
as
select orderid, productid, unitprice, quantity, discount,
       cast(unitprice * quantity * (1-discount) as number(10,2)) as value
from orderdetails;
```

```sql
select * from vw_order_details
where orderid = 10250;
```

wynik:

	orderid	productid	unitprice	quantity	discount	value
1	10250	41	7.7	10	0	77.00
2	10250	51	42.4	35	0.15	1261.40
3	10250	65	16.8	15	0.15	214.20

Widoki c.d.

```sql
create or replace view vw_order_total_1
as
select orderid, cast(sum(unitprice * quantity * (1-discount)) as decimal(10,2)) as total
from orderdetails
group by orderid;
```

```sql
create or alter view vw_order_total_2
as
select orderid, sum(value) as total
from vw_order_details
group by orderid;
```

wynik (fragment):

	orderid	total
1	10248	440.00
2	10249	1863.40
3	10250	1552.60
4	10251	654.06

Widoki c.d.

```
create or alter view vw_order_total_3
as
select o.orderid, cast(o.orderdate as date) orderdate, o.customerid, c.companyname,
       sum(value) as total
from vw_order_details od join orders o on od.orderid = o.orderid
join customers c on o.customerid = c.customerid
group by o.orderid, o.orderdate,  o.customerid, c.companyname;
```

wynik (fragment):

	orderid	orderdate	customerid	companyname	total
1	10248	1996-07-04	VINET	Vins et alcools Chevalier	440
2	10249	1996-07-05	TOMSP	Toms Spezialitäten	1863.4
3	10250	1996-07-08	HANAR	Hanari Carnes	1552.6
4	10251	1996-07-08	VICTE	Victuailles en stock	654.06
5	10252	1996-07-09	SUPRD	Suprêmes délices	3597.9

Funkcje, Procedury

```
FUNCTION name

IS

BEGIN

  RETURN value

[EXCEPTION]

END;
```

```
PROCEDURE name

IS

BEGIN

[EXCEPTION]

END;
```

Funkcja zwracająca wartość

```
create or replace function f_hello(name varchar)
    return varchar
as
    hello varchar(50) := 'hello';
    result varchar(50);
begin
    if name is null then
        raise_application_error(-20001, 'empty name');
    end if;

    result :=  hello || ' ' ||  name;
    return result;
end;
```

```
select f_hello('jan') from dual;

select s.*, f_hello(companyname) from shippers s;
```

```
declare
    v varchar(50);
begin
   select f_hello('jan') into v from dual;
   dbms_output.put_line(v);
end;
```

Funkcja zwracająca tabelę

- Definicja typu
  - obiekt
  - tablica obiektów
- Definicja funkcji
  - `select … bulk collect into ….`
  - `loop … pipe row`

Funkcja zwracająca tabelę - przykład

- przykładowe polecenie

```
select orderid, orderdate, total
from vw_order_total_3
where customerid = 'ALFKI';
```

Definicja typu

```
create or replace type customer_order_total as OBJECT
(
    orderid        int,
    customerid     varchar(5),
    orderdate      date,
    total          number(10,2)
);
```

```
create or replace type customer_order_total_table is table of customer_order_total;
```

Definicja funkcji

```
create or replace function f_customer_order_total(customerid varchar)
    return customer_order_total_table
as
    result customer_order_total_table;
begin
    select customer_order_total(ot.orderid, ot.customerid, ot.orderdate, ot.total)
    bulk collect
    into result
    from vw_order_total_3 ot
    where ot.customerid = f_customer_order_total.customerid;

    return result;
end;
```

Wywołanie funkcji

```sql
select orderid, orderdate, total from table(f_customer_order_total ('ALFKI'));

select * from f_customer_order_total ('ALFKI');

select f.orderid, f.orderdate, f.total, c.companyname, c.phone
from f_customer_order_total('ALFKI') f
join customers c on f.customerid = c.customerid;

select sum(total) from f_customer_order_total('ALFKI');
```

Wywołanie funkcji

```
select orderid, orderdate, total from table(f_customer_order_total ('ALFKI'));

select * from table(f_customer_order_total ('ALFKI'));
```

	ORDERID	CUSTOMERID	ORDERDATE	TOTAL
1	10643	ALFKI	1997-08-25	814.50
2	10692	ALFKI	1997-10-03	878.00
3	10702	ALFKI	1997-10-13	330.00
4	10835	ALFKI	1998-01-15	845.80
5	10952	ALFKI	1998-03-16	471.20

```
select * from f_customer_order_total ('PARIS');

select * from f_customer_order_total ('ala');
```

- zbiór pusty

Kontrola argumentów

```
create or replace function f_customer_order_total(customerid varchar)
    return customer_order_total_table
as
    result customer_order_total_table;
    valid int;
begin
    select count(*) into valid
    from customers c
    where c.customerid = f_customer_order_total.customerid;

    if valid = 0 then
        raise_application_error(-20001, 'customer not found');
    end if;

    select customer_order_total(ot.orderid, ot.customerid, ot.orderdate, ot.total)
    bulk collect
    into result
    from vw_order_total_3 ot
    where ot.customerid = f_customer_order_total.customerid;

    return result;
end;
```

Kontrola argumentów c.d.

```
select * from table(f_customer_order_total ('ALFKI'));
```

	ORDERID	CUSTOMERID	ORDERDATE	TOTAL
1	10643	ALFKI	1997-08-25	814.50
2	10692	ALFKI	1997-10-03	878.00
3	10702	ALFKI	1997-10-13	330.00
4	10835	ALFKI	1998-01-15	845.80
5	10952	ALFKI	1998-03-16	471.20

```
select * from f_customer_order_total ('PARIS');
```

- zbiór pusty

```
select * from f_customer_order_total ('ala');
```

> Error
> [72000][20001] ORA-20001: customer not found
> ORA-06512: at "NORTHWIND_M.F_CUSTOMER_ORDER_TOTAL", line 12

Kontrola argumentów – funkcja pomocnicza

```
create or replace function f_customer_exist(c_id in customers.customerid%type)
    return boolean
as
    exist number;
begin
    select case
                when exists(select * from customers where customerid = c_id) then 1
                else 0
            end
    into exist from dual;

    if exist = 1 then
        return true;
    else
        return false;
    end if;
end;
```

Kontrola argumentów c.d.

- wykorzystanie funkcji pomocniczej

```
create or replace function f_customer_order_total(customerid varchar)
    return customer_order_total_table
as
    result customer_order_total_table;
    valid int;
begin
    if not f_customer_exist(customerid) then
         raise_application_error(-20001, 'customer not found');
    end if;

    select customer_order_total(ot.orderid, ot.customerid, ot.orderdate, ot.total)
    bulk collect
    into result
    from vw_order_total_3 ot
    where ot.customerid = f_customer_order_total.customerid;

    return result;
end;
```

Kontrola argumentów c.d.

- jeszcze inny sposób

```
create or replace function f_customer_order_total(customerid varchar)
    return customer_order_total_table
as
    result customer_order_total_table;
    tmp char(1);
begin
    select 1 into tmp from customers c where c.customerid = f_customer_order_total.customerid;

    select customer_order_total(ot.orderid, ot.customerid, ot.orderdate, ot.total)
    bulk collect
    into result
    from vw_order_total_3 ot
    where ot.customerid = f_customer_order_total.customerid;

    return result;
exception
    when NO_DATA_FOUND then
        raise_application_error(-20001, 'customer not found');
end;
```

Kontrola argumentów – procedura pomocnicza

```
create or replace procedure p_customer_exist(c_id in customers.customerid%type)
as
    exist number;
begin
    select case
                when exists(select * from customers where customerid = c_id) then 1
                else 0
            end
    into exist from dual;
    if exist <> 1 then
       raise_application_error(-20001, 'customer not found');
    end if;
end;
```

Kontrola argumentów c.d.

- wykorzystanie procedury

```
create or replace function f_customer_order_total(customerid varchar)
    return customer_order_total_table
as
    result customer_order_total_table;
begin
    p_customer_exist(customerid);

    select customer_order_total(ot.orderid, ot.customerid, ot.orderdate, ot.total)
    bulk collect
    into result
    from vw_order_total_3 ot
    where ot.customerid = f_customer_order_total.customerid;

    return result;
end;
```

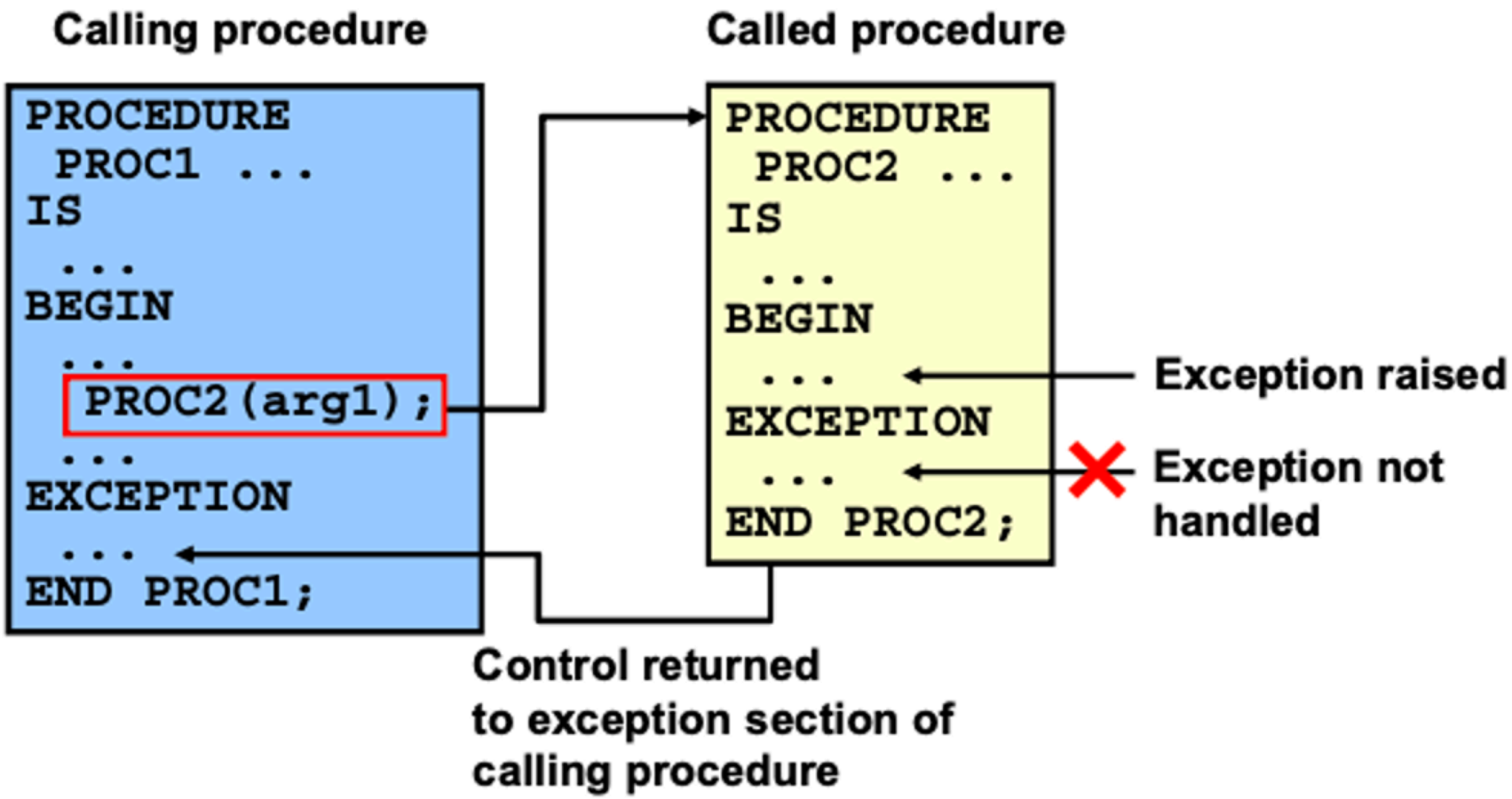
Kontrola argumentów – procedura pomocnicza

- inna wersja procedury

```
create or replace procedure p_customer_exist(c_id in customers.customerid%type)
as
    tmp char(1);
begin
    select 1 into tmp from customers c where c.customerid = c_id;

exception
    when NO_DATA_FOUND then
        raise_application_error(-20001, 'customer not found !!!');
end;
```

Wyjątki


Calling procedure
PROCEDURE
PROC1 ...
IS
...
BEGIN
...
PROC2(arg1);
...
EXCEPTION
...
END PROC1;
Called procedure
PROCEDURE
PROC2 ...
IS
...
BEGIN
...
EXCEPTION
...
END PROC2;
Exception raised
Exception not handled
Control returned to exception section of calling procedure

Definicja funkcji - inna metoda

```
create or replace function f_customer_order_total(customerid varchar)
    return customer_order_total_table pipelined
as
begin
        for v_row in ( select ot.orderid, ot.customerid, ot.orderdate, ot.total
                       from vw_order_total_3 ot
                       where ot.customerid = f_customer_order_total.customerid)
        Loop
            pipe row (customer_order_total(v_row.orderid, v_row.customerid,
                                           v_row.orderdate, v_row.total));
        end loop;

    return;
end;
```

- może być wydajniejsza w przypadku jeśli funkcja zwraca wiele wierszy

Definicja funkcji - kolejny przykład

```
create or replace function f_customer_order_total_2(customerid varchar, start_date date, end_date date)
    return customer_order_total_table
as
    result customer_order_total_table;
begin
    p_customer_exist(customerid);

    select customer_order_total(ot.orderid, ot.customerid, ot.orderdate, ot.total)
    bulk collect
    into result
    from vw_order_total_3 ot
    where ot.customerid = f_customer_order_total_2.customerid
          and ot.orderdate >= start_date and ot.orderdate <= end_date;

    return result;
end;
```

Procedury - modyfikacja danych

```
create or replace procedure p_add_order(vcustomerid varchar, vemployeeid int,
                                        vrequireddate date)
as
  vorderdate date;
begin
    vorderdate := trunc(sysdate);
    insert into orders(customerid,employeeid,orderdate,requireddate)
    values(vcustomerid,vemployeeid,vorderdate,vrequireddate);
end;
```

- wywołanie

```
declare
   requiredate date := trunc(sysdate) + 7;
begin
    p_add_order('ALFKI',1, requiredate);
    commit;
end;
```

- OK

Procedury - modyfikacja danych c.d.

```oracle_plsql
declare
   requiredate date := trunc(sysdate) + 7;
begin
    p_add_order('ala',1, requiredate);
    commit;
end;
```

> Error
> [23000][2291]
> ORA-02291: integrity constraint (NORTHWIND_M.FK_ORDERS_CUSTOMERS) violated - parent key not found
> ORA-06512: at "NORTHWIND_M.P_ADD_ORDER", line 6

Kontrola argumentów

```
create or replace procedure p_add_order(vcustomerid varchar, vemployeeid int,
                                        vrequireddate date)
as
  vorderdate date;
begin

    p_customer_exist(vcustomerid);

    vorderdate := trunc(sysdate);
    insert into orders(customerid,employeeid,orderdate,requireddate)
    values(vcustomerid,vemployeeid,vorderdate,vrequireddate);
end;
```

```
declare
   requiredate date := trunc(sysdate) + 7;
begin
    p_add_order('ala',1, requiredate);
    commit;
end;
```

> Error
> [72000][20001]
> ORA-20001: customer not found !!!
> ORA-06512: at "NORTHWIND_M.P_CUSTOMER_EXIST", line 9
> ORA-06512: at "NORTHWIND_M.P_ADD_ORDER", line 6

Procedury - modyfikacja danych c.d.

```
create or replace procedure p_add_detail(vorderid int, vproductid int,
                                         vquantity int, vdiscount decimal(3,2))
as
    declare vunitprice decimal(10,2);
begin
    select unitprice into vunitprice from products where productid = vproductid;

    insert into orderdetails(orderid, productid, unitprice, quantity, discount)
    values(vorderid, vproductid, vunitprice, vquantity, vdiscount);
end;
```

```
begin
    p_add_detail(20000, 1,3, 0.12);
    commit;
end;
```

```
declare
   requiredate date := trunc(sysdate) + 7;
   vorderid int;
begin
    p_add_order('ALFKI',1, requiredate);
    p_add_detail(s_orders_seq.currval, 1,3, 0.12);
    commit;
end;
```

Procedury - modyfikacja danych c.d.

- próba "sprzedaży" nieistniejącego produktu

```
declare
   requiredate date := trunc(sysdate) + 7;
   vorderid int;
begin
    p_add_order('ALFKI',1, requiredate);
    p_add_detail(s_orders_seq.currval, 99,3, 0.12);
    commit;
end;
```

> Error
> [02000][1403]
> ORA-01403: no data found
> ORA-06512: at "NORTHWIND_M.P_ADD_DETAIL", line 6 ORA-06512: at line 6

Procedury - modyfikacja danych c.d.

```oracle
create or replace procedure p_add_detail(vorderid int, vproductid int,
                                         vquantity int, vdiscount number)
as
    vunitprice number(10,2);
begin
    select unitprice into vunitprice from products where productid = vproductid;

    insert into orderdetails(orderid, productid, unitprice, quantity, discount)
    values(vorderid, vproductid, vunitprice, vquantity, vdiscount);
exception
    when NO_DATA_FOUND then
        raise_application_error(-20002, 'product not found');
end;
```

```oracle
declare
   requiredate date := trunc(sysdate) + 7;
   vorderid int;
begin
    p_add_order('ALFKI',1, requiredate);
    p_add_detail(s_orders_seq.currval, 99,3, 0.12);
    commit;
end;
```

> Error
> [72000][20001]
> ORA-20001: product not found
> ORA-06512: at "NORTHWIND_M.P_ADD_DETAIL", line 12 ORA-06512: at line 6

Procedury - modyfikacja danych c.d.

- Oczywiście procedura wymaga jeszcze modyfikacji
  - produkt może istnieć, ale może być wycofany
  - produktu może nie być w magazynie
  - po dodaniu produktu do magazynu należało by zmniejszyć stan magazynu

Triggery (wyzwalacze)

Uruchamiane przez zajście określonego zdarzenia w bazie danych

- np. modyfikacji danych
  - insert, update, delete
- Cele stosowania
  - automatyzacja operacji w bazie danych
  - wymuszanie złożonych reguł biznesowych
  - kontrola złożonych warunków integralnościowych
  - śledzenie działań użytkowników
    - modyfikacji danych
  - modyfikacja za pomocą widoków

Triggery c.d.

```
TRIGGER name

  <moment uruchomienia>
  <zdarzenie uruchamiające> ON { relacja/tabela | widok }
  [ WHEN warunek ]

  [ FOR EACH ROW ]

[ DECLARE <deklaracje > ]

BEGIN
   ...
END;
```

Triggery c.d.

- Zdarzenie uruchamiające:
  - polecenie DML
    - INSERT, UPDATE, DELETE,
  - polecenie DDL
    - CREATE, ALTER,
  - zdarzenie w bazie danych:
    - np. zalogowanie/wylogowanie użytkownika, błąd, uruchomienie/zatrzymanie bazy danych.
- Moment uruchomienia
  - BEFORE
  - AFTER
  - INSTEAD OFF

Triggery c.d.

- WHEN warunek
  - warunek determinujący wykonanie triggera
- FOR EACH ROW
  - trigger "wierszowy"
- Odwołanie do wartości atrybutów modyfikowanej tabeli
  - `:OLD.nazwa_atrybutu`
    - wartość sprzed wykonania polecenia
  - `:NEW.nazwa_atrybutu`
    - wartość po wykonaniu polecenia

Triggery c.d.

- Compound trigger

```
create or replace trigger compound_trigger_name
for [insert|delete]update [of column] on table_name
compound trigger
   -- deklaracje
     before statement is
     begin
        ...
     end before statement;

     before each row is
     begin
      -- ...
     end before each row;

     after each row is
     begin
       ...
     end after each row;

     after statement is
     begin
       ...
     end after statement;

end compound_trigger_name;
```

Triggery - przykład

- przykladowa tabela

```
create sequence s_test_seq
   start with 1
   increment by 1;

create table test (
    id int primary key,
    val1 int,
    val2 varchar(10)
);

alter table test
    modify id int default stest_seq.nextval;
```

Triggery - przykład

- trigger wierszowy

```
create or replace trigger tr_test
    after insert or update or delete
    on test
    for each row
begin
    dbms_output.put_line('trigger test, id: ' || :NEW.id ||
                         ', old_val1: ' || :OLD.val1 ||
                         ', new_val1: ' || :NEW.val1);
end;
```

```
insert into test (val1, val2)
values (1,'1');

insert into test (val1, val2)
values (2,'2');

update test
set val1 = val1 + 1
where 1 = 1;

delete test
where 1 = 1
```

Triggery - przykład c.d.

- fraza `when`

```
create or replace trigger tr_test
    after insert or update or delete
    on test
    for each row
    when NEW.val1 > 3
begin
    dbms_output.put_line('trigger test, id: ' || :NEW.id ||
                         ', old_val1: ' || :OLD.val1 ||
                         ', new_val1: ' || :NEW.val1);
end;
```

```
insert into test (val1, val2)
values (1,'1');

insert into test (val1, val2)
values (2,'2');

update test
set val1 = val1 + 1
where 1 = 1;

delete test
where 1 = 1
```