Bazy danych

SQL, Programowanie proceduralne, PL/SQL

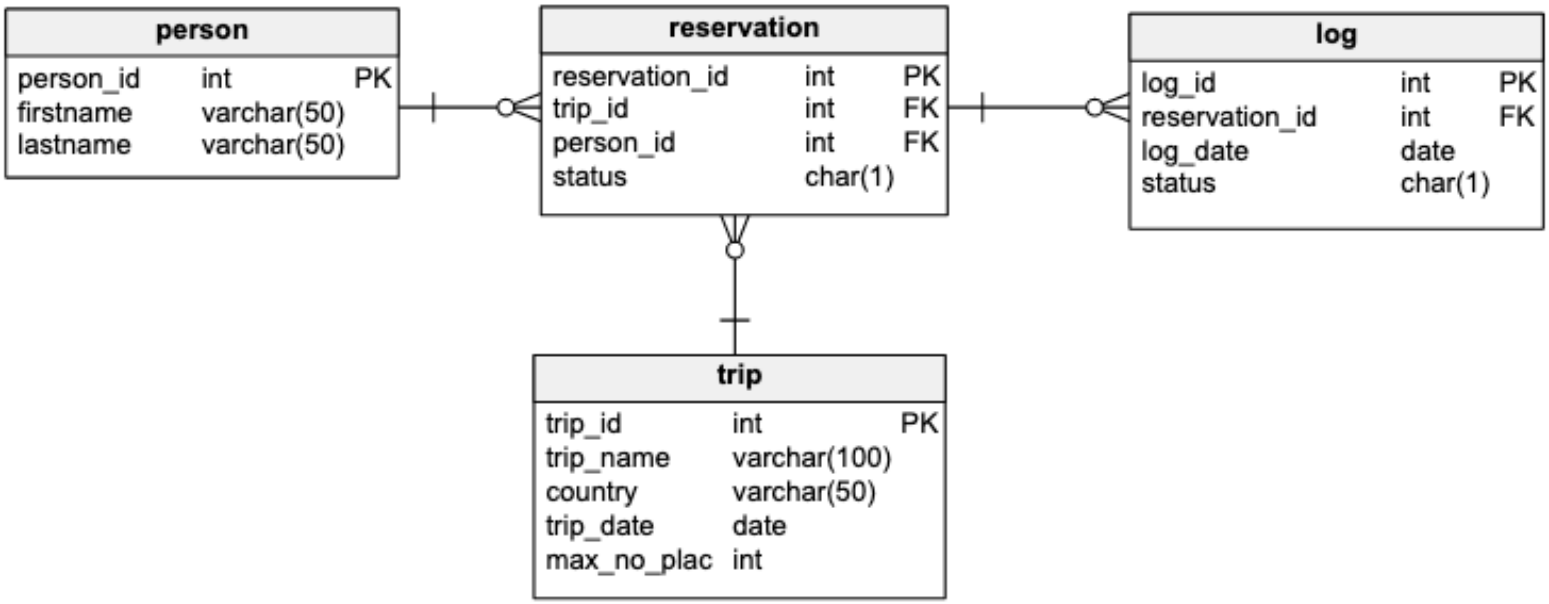
Programowanie proceduralne

- SQL
  - deklaratywny
- Rozszerzenie SQL o elementy programowania proceduralnego
  - zmienne, stałe
  - struktury sterujące
  - procedury składowane
  - funkcje
  - kursory
  - obsługa błędów, wyjątków

SZBD, Języki

- MS SQL Server
  - Transact SQL, T-SQL
- Oracle
  - PL/SQL
- PostgreSQL
  - PL/pgSQL

Przykład


person
person_id int PK
firstname varchar(50)
lastname varchar(50)
reservation
reservation_id int PK
trip_id int FK
person_id int FK
status char(1)
log
log_id int PK
reservation_id int FK
log_date date
status char(1)
trip
trip_id int PK
trip_name varchar(100)
country varchar(50)
trip_date date
max_no_plac int

Przykład - Oracle

```
create table person
(
  person_id int generated always as identity not null,
  firstname varchar(50),
  lastname varchar(50),
  constraint person_pk primary key ( person_id ) enable
);

create table trip
(
  trip_id int generated always as identity not null,
  trip_name varchar(100),
  country varchar(50),
  trip_date date,
  max_no_places int,
  constraint trip_pk primary key ( trip_id ) enable
);
```

Przykład - Oracle

```
create table reservation
(
  reservation_id int generated always as identity not null,
  trip_id int,
  person_id int,
  status char(1),
  constraint reservation_pk primary key ( reservation_id ) enable
);

alter table reservation
add constraint reservation_fk1 foreign key
( person_id ) references person ( person_id ) enable;

alter table reservation
add constraint reservation_fk2 foreign key
( trip_id ) references trip ( trip_id ) enable;

alter table reservation
add constraint reservation_chk1 check
(status in ('N','P','C')) enable;
```

Przykład - Oracle

```
create table log
(
   log_id int  generated always as identity not null,
   reservation_id int not null,
   log_date date  not null,
   status char(1),
   constraint log_pk primary key ( log_id ) enable
);

alter table log
add constraint log_chk1 check
(status in ('N','P','C')) enable;

alter table log
add constraint log_fk1 foreign key
( reservation_id ) references reservation ( reservation_id ) enable;
```

Oracle PL/SQL

- Koncepcja języka PL/SQL
- Struktura bloku PL/SQL
- Deklarowanie zmiennych i stałych
- Przegląd podstawowych konstrukcji sterujących języka PL/SQL
- 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;
```

Typ i zmienna rekordowa

```
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

```
declare
	f person.firstname%TYPE;
begin
	f := 'jan';
	dbms_output.Put_line(f);
end;

declare
	f person%ROWTYPE;
begin
	f.firstname := 'jan';
	dbms_output.Put_line('firstname: ' || f.firstname);
end;
```

%TYPE %ROWTYPE

```
declare
	f person.firstname%TYPE;
begin
	f := 'jan';
	dbms_output.Put_line(f);
end;

declare
	f person%ROWTYPE;
begin
	f.firstname := 'jan';
	dbms_output.Put_line('firstname: ' || f.firstname);
end;
```

Instrukcje sterujące

- IF
- CASE
- LOOP
  - WHILE
  - FOR

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

SELECT … INTO

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


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

INSERT

```
declare
	first varchar2(50); last varchar2(50); id int;
begin
	insert into person(firstname, lastname)
	values ('tom', 'smith')
	returning person_id into id;

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

declare
	p person%rowtype;
begin
	insert into person(firstname, lastname)
	values ('adam', 'smith')
	returning person_id, firstname into p.person_id, p.firstname;

	dbms_output.Put_line('id: ' || p.person_id || ' firstname: ' || p.firstname);
end;
```

UPDATE

```
declare
        p person%rowtype;
begin
        p.person_id := 1;
        p.firstname := 'Jan Jan';
        p.lastname := 'Nowak Nowak';

        update person
        set row = p
        where person_id = 1 ;
end;
```

Wyjątki c.d. - przykład

```
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. - przykład

```
declare
	i int;
	exc1 exception;
begin
	insert into test1(tname, status)
	values ('ala', 'A');

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

	dbms_output.put_line('OK');

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

Widoki

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

View

```
create view trip_redervation
as
	select r.reservation_id, r.trip_id, p.person_id,
		p.firstname, p.lastname
	from reservation r join person p
		on r.person_id = p.person_id
	where r.trip_id = 1;


select * from trip_reservation;
```

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('') from dual;

select p.*, f_hello(lastname) from person p;
```

Funkcja zwracająca tabelę

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

Przykład

```
select r.reservation_id, r.trip_id, p.person_id,
        p.firstname, p.lastname
from reservation r  join person p
       on r.person_id = p.person_id
where r.trip_id = 1;
```

Definicja typu

```
create or replace type trip_participant as OBJECT
(
   reservation_id    int,
   trip_id           int,
   person_id         int,
   firstname         varchar2(50),
   Lastname          varchar2(50)
);


create or replace type trip_participant_table is table of trip_participant;
```

Definicja funkcji

```
create or replace function f_trip_participants(trip_id int)
        return trip_participant_table
as
        result trip_participant_table;
begin
        select trip_participant(r.reservation_id, r.trip_id, p.person_id,
                        p.firstname, p.lastname) bulk collect
        into result
        from reservation r  join person p
                on r.person_id = p.person_id
        where r.trip_id = f_trip_participants.trip_id;
        return result;
end;

select * from f_trip_participants (1) ;
select * from table(f_trip_participants (1)) ;
```

Kontrola argumentów

```
create or replace function f_trip_participants1(trip_id int)
        return trip_participant_table
as
        result trip_participant_table;
        valid  int;
begin
        select count(*) into valid
        from trip t
        where t.trip_id = f_trip_participants1.trip_id;

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

        select trip_participant(r.reservation_id, r.trip_id, p.person_id,
                                p.firstname, p.lastname)
        bulk collect into result
        from reservation r  join person p
              on r.person_id = p.person_id
        where r.trip_id = f_trip_participants1.trip_id;

        return result;
end;
```

Kontrola argumentów – funkcja pomocnicza

```
create or replace function trip_exist(t_id in trip.trip_id%type)
	return boolean
as
	exist number;
begin
	select case
			when exists(select * from trip where trip_id = t_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.

```
create or replace function f_trip_participants2(trip_id int)
        return trip_participant_table
as
        result trip_participant_table;
begin
        if not trip_exist(trip_id) then
                raise_application_error(-20001, 'trip not found');
        end if;

        select trip_participant(r.reservation_id, r.trip_id, p.person_id,
                        p.firstname, p.lastname)
        bulk collect into result
        from reservation r join person p
                on r.person_id = p.person_id
        where r.trip_id = f_trip_participants2.trip_id;

        return result;
end;
```

Kontrola argumentów c.d.

```
create or replace function f_trip_participants4(trip_id int)
        return trip_participant_table
as
        result trip_participant_table;
        tmp char(1);
begin

        select 1  into tmp from trip where trip_id = f_trip_participants4.trip_id;

        select trip_participant(r.reservation_id, r.trip_id, p.person_id,
                        p.firstname, p.lastname)
        bulk collect into result
        from reservation r join person p
              on r.person_id = p.person_id
        where r.trip_id = f_trip_participants4.trip_id;

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

Kontrola argumentów c.d.

```
create or replace procedure trip_exist5(t_id trip.trip_id%type)
as
	tmp char(1);
begin
	select 1  into tmp from trip where trip_id = t_id;

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

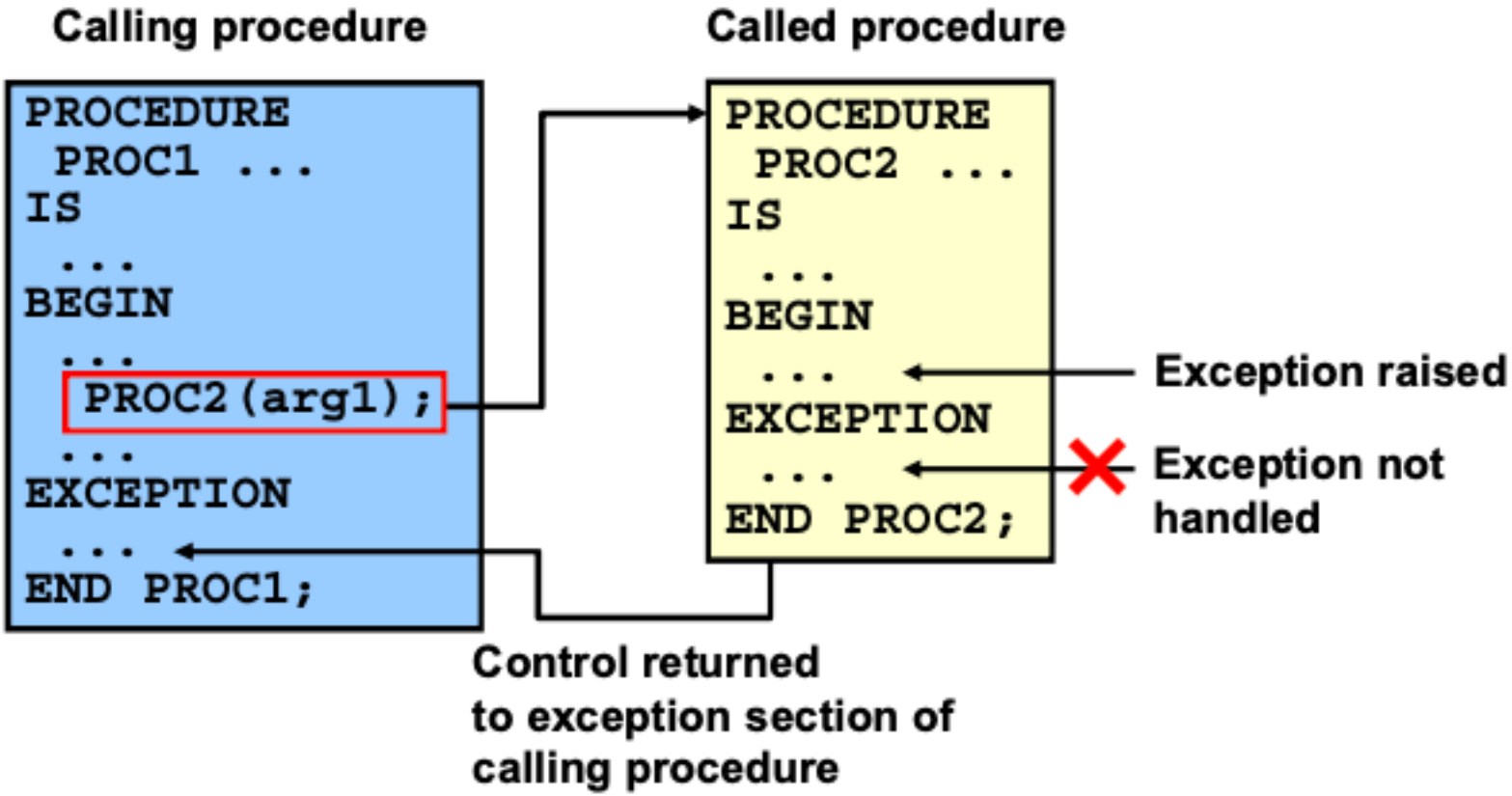
Kontrola argumentów c.d.

```
create or replace function f_trip_participants5(trip_id int)
        return trip_participant_table
as
        result trip_participant_table;
begin
        trip_exist5(trip_id);

        select trip_participant(r.reservation_id, r.trip_id, p.person_id,
                        p.firstname, p.lastname)
        bulk collect into result
        from reservation r join person p
                on r.person_id = p.person_id
        where r.trip_id = f_trip_participants5.trip_id;

        return result;
end;
```

Wyjątki


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

Funkcja – inna metoda

```
create or replace function f_trip_participants6(p_trip_id int)
   return trip_participant_table pipelined
as
begin
        for v_row in (select r.reservation_id,
                             r.trip_id,
                             p.person_id,
                             p.firstname,
                             p.lastname
                        from reservation r  join person p
                                    on r.person_id = p.person_id
                         where r.trip_id = p_trip_id)
        Loop
                pipe row (trip_participant(v_row.reservation_id, v_row.trip_id,
                                           v_row.person_id, v_row.firstname, v_row.lastname));
        end loop;

        return;
end;
```

Procedura

```
create or replace procedure modify_reservation_status(p_reservation_id int, p_status char)
as
        tmp char(1);
begin
        select 1 into tmp from reservation where reservation_id = p_reservation_id;

        if p_status not in ('N', 'P', 'C') then
                raise_application_error(-20002, 'wrong status');
        end if;

        update reservation
        set status = p_status
        where reservation_id = p_reservation_id;

exception
        when NO_DATA_FOUND then
                raise_application_error(-20001, 'reservation not found');
end;


begin
        modify_reservation_status (  1, 'C');
        commit;
end;
```

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

TRIGGER name
    <moment uruchomienia>
    <zdarzenie uruchamiające> ON { relacja | perspektywa }
    [WHEN warunek]
    [FOR EACH ROW]

[**DECLARE** <deklaracje >]

BEGIN

END;

Ttriggery

- 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 OF

Trigger

```
create or replace trigger tr_update_log
        after insert or update
        on reservation
        for each row
begin
        insert into log (reservation_id, log_date, status, no_places)
        values (:new.reservation_id, current_date, :new.status, :new.no_places);
end;
```

Trigger 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

Ćwiczenie – lab.

- Oprogramowanie operacji w przykładowej bazie danych