**ORACLE PL/SQL**

* **BLOCKS:**

SET SERVEROUTPUT ON; **--- used to display the output**

DECLARE

BEGIN

dbms\_output.put\_line('Hello World'); ---- print something

BEGIN

dbms\_output.put\_line('RIYA'); ---- nested blocks

END;

END;

* **VARIABLES**
* SET SERVEROUTPUT ON;

DECLARE

v varchar2(20) := 2 + 25 \* 3;

BEGIN

dbms\_output.put\_line(v);

END;

* DECLARE

v\_text varchar2(50) NOT NULL DEFAULT 'Hello';

v\_number1 number := 50;

v\_number2 number(2) := 50.42;

v\_number3 number(10,2) := 50.42;

v\_number4 PLS\_INTEGER := 50;

v\_number5 BINARY\_float := 50.42;

v\_DATE1 DATE := '22-NOV-18 12:01:32';

v\_DATE2 timestamp := systimestamp;

v\_DATE3 timestamp(9) WITH TIME ZONE := systimestamp;

v\_DATE4 interval day(4) to second (3) := '124 02:05:21.012 ';

v\_DATE5 interval year to month := '12-3';

BEGIN

V\_TEXT := 'PL/SQL' || 'Course';

DBMS\_OUTPUT.PUT\_LINE(V\_TEXT);

DBMS\_OUTPUT.PUT\_LINE(v\_number1);

DBMS\_OUTPUT.PUT\_LINE(v\_number2);

DBMS\_OUTPUT.PUT\_LINE(v\_number3);

DBMS\_OUTPUT.PUT\_LINE(v\_number4);

DBMS\_OUTPUT.PUT\_LINE(v\_number5);

DBMS\_OUTPUT.PUT\_LINE(v\_DATE1);

DBMS\_OUTPUT.PUT\_LINE(v\_DATE2);

DBMS\_OUTPUT.PUT\_LINE(v\_DATE3);

DBMS\_OUTPUT.PUT\_LINE(v\_DATE4);

DBMS\_OUTPUT.PUT\_LINE(v\_DATE5);

END;

* DECLARE

v\_boolean boolean := true;

BEGIN

dbms\_output.put\_line(sys.diutil.bool\_to\_int(v\_boolean));

END;

* **%TYPE ATTRIBUTE**:
  + It is used if we want to assign a type to the variable which is same as another variable, mostly used in database column
  + To get the type of a database row, we use the %rowtype attribute

desc employees;

declare

V\_TYPE employees.JOB\_ID%TYPE;

V\_TYPE2 V\_TYPE%TYPE;

V\_TYPE3 employees.JOB\_ID%TYPE ;

begin

v\_type := 'IT\_PROG';

v\_type2 := 'SA\_MAN';

v\_type3 := NULL;

dbms\_output.put\_line(v\_type);

dbms\_output.put\_line(v\_type2);

dbms\_output.put\_line('HELLO' || v\_type3);

end;

* **DELIMITERS AND COMMENTING**

DECLARE

V\_TEXT VARCHAR2(10):= 'PL/SQL';

BEGIN

--This is a single line comment

/\* This is a

multi line

comment \*/

--DBMS\_OUTPUT.PUT\_LINE(V\_TEXT || ' is a good language');

null;

END;

* **VARIABLE SCOPE**

begin <<outer>>

DECLARE

--v\_outer VARCHAR2(50) := 'Outer Variable!';

v\_text VARCHAR2(20) := 'Out-text';

BEGIN

DECLARE

v\_text VARCHAR2(20) := 'In-text';

v\_inner VARCHAR2(30) := 'Inner Variable';

BEGIN

--dbms\_output.put\_line('inside -> ' || v\_outer);

--dbms\_output.put\_line('inside -> ' || v\_inner);

dbms\_output.put\_line('inner -> ' || v\_text);

dbms\_output.put\_line('outer -> ' || outer.v\_text);

END;

--dbms\_output.put\_line('inside -> ' || v\_inner);

--dbms\_output.put\_line(v\_outer);

dbms\_output.put\_line(v\_text);

END;

END outer;

* **BIND VARIABLES**
* NOTE: When you run a bind variable creation and select statement together, SQL Developer may return an error. But when you execute them separately, there will be no problem
* These variables are no block variables, also we can directly print them
* set serveroutput on;

set autoprint on; --- prints all the bind variables that are executed

variable var\_text varchar2(30); **--declaring bind variables**

variable var\_number NUMBER;

variable var\_date DATE; **-- this type can not be used bind varaiables**

declare

v\_text varchar2(30);

begin

:var\_text := 'Hello SQL'; **--- assigning and accessing it**

:var\_number := 20;

v\_text := :var\_text;

--dbms\_output.put\_line(v\_text);

--dbms\_output.put\_line(:var\_text);

end;

print var\_text;

variable var\_sql number;

* begin

:var\_sql := 100;

end;

select \* from employees where employee\_id = :var\_sql;

* **IF STATEMENTS**

set serveroutput on;

declare

v\_number number := 30;

begin

if v\_number < 10 then

dbms\_output.put\_line('I am smaller than 10');

elsif v\_number < 20 then

dbms\_output.put\_line('I am smaller than 20');

elsif v\_number < 30 then

dbms\_output.put\_line('I am smaller than 30');

else

dbms\_output.put\_line('I am equal or greater than 30');

end if;

end;

* declare

v\_number number := 5;

v\_name varchar2(30) := 'Adam';

begin

if v\_number < 10 or v\_name = 'Carol' then

dbms\_output.put\_line('HI');

dbms\_output.put\_line('I am smaller than 10');

elsif v\_number < 20 then

dbms\_output.put\_line('I am smaller than 20');

elsif v\_number < 30 then

dbms\_output.put\_line('I am smaller than 30');

else

if v\_number is null then

dbms\_output.put\_line('The number is null..');

else

dbms\_output.put\_line('I am equal or greater than 30');

end if;

end if;

end;

* **CASE EXPRESSIONS**
* declare

v\_job\_code varchar2(10) := 'SA\_MAN';

v\_salary\_increase number;

begin

v\_salary\_increase := case v\_job\_code

when 'SA\_MAN' then 0.2

when 'SA\_REP' then 0.3

else 0

end;

dbms\_output.put\_line('Your salary increase is : '|| v\_salary\_increase);

end;

* **SEARCHED CASE EXPRESSION**

declare

v\_job\_code varchar2(10) := 'IT\_PROG';

v\_department varchar2(10) := 'IT';

v\_salary\_increase number;

begin

v\_salary\_increase := case

when v\_job\_code = 'SA\_MAN' then 0.2

when v\_department = 'IT' and v\_job\_code = 'IT\_PROG' then 0.3

else 0

end;

dbms\_output.put\_line('Your salary increase is : '|| v\_salary\_increase);

end;

* **CASE STATEMENTS**

declare

v\_job\_code varchar2(10) := 'IT\_PROG';

v\_department varchar2(10) := 'IT';

v\_salary\_increase number;

begin

case

when v\_job\_code = 'SA\_MAN' then

v\_salary\_increase := 0.2;

dbms\_output.put\_line('The salary increase for a Sales Manager is : '|| v\_salary\_increase);

when v\_department = 'IT' and v\_job\_code = 'IT\_PROG' then

v\_salary\_increase := 0.2;

dbms\_output.put\_line('The salary increase for a Sales Manager is : '|| v\_salary\_increase);

else

v\_salary\_increase := 0;

dbms\_output.put\_line('The salary increase for this job code is : '|| v\_salary\_increase);

end CASE;

end;

* **BASIC LOOPS**

declare

v\_counter number(2) := 1;

begin

loop

dbms\_output.put\_line('My counter is : '|| v\_counter);

v\_counter := v\_counter + 1;

--if v\_counter = 10 then

-- dbms\_output.put\_line('Now I reached : '|| v\_counter);

-- exit;

--end if;

exit when v\_counter > 10;

end loop;

end;

* **WHILE LOOPS**

declare

v\_counter number(2) := 1;

begin

while v\_counter <= 10 loop

dbms\_output.put\_line('My counter is : '|| v\_counter);

v\_counter := v\_counter + 1;

-- exit when v\_counter > 3;

end loop;

end;

* **FOR LOOPS**

begin

for i in REVERSE 1..3 loop

dbms\_output.put\_line('My counter is : '|| i);

end loop;

end;

* **NESTED LOOPS**

declare

v\_inner number := 1;

begin

for v\_outer in 1..5 loop

dbms\_output.put\_line('My outer value is : ' || v\_outer );

v\_inner := 1;

loop

v\_inner := v\_inner+1;

dbms\_output.put\_line(' My inner value is : ' || v\_inner );

exit when v\_inner\*v\_outer >= 15;

end loop;

end loop;

end;

* **NESTED LOOPS WITH LABELS**

declare

v\_inner number := 1;

begin

**<<outer\_loop>> --labelling**

for v\_outer in 1..5 loop

dbms\_output.put\_line('My outer value is : ' || v\_outer );

v\_inner := 1;

**<<inner\_loop>> --labelling**

loop

v\_inner := v\_inner+1;

dbms\_output.put\_line(' My inner value is : ' || v\_inner );

exit outer\_loop when v\_inner\*v\_outer >= 16;

exit when v\_inner\*v\_outer >= 15;

end loop inner\_loop;

end loop outer\_loop;

end;

* **CONTINUE STATEMENT**
* declare

v\_inner number := 1;

begin

for v\_outer in 1..10 loop

dbms\_output.put\_line('My outer value is : ' || v\_outer );

v\_inner := 1;

while v\_inner\*v\_outer < 15 loop

v\_inner := v\_inner+1;

continue when mod(v\_inner\*v\_outer,3) = 0;

dbms\_output.put\_line(' My inner value is : ' || v\_inner );

end loop;

end loop;

end;

* declare

v\_inner number := 1;

begin

<<outer\_loop>>

for v\_outer in 1..10 loop

dbms\_output.put\_line('My outer value is : ' || v\_outer );

v\_inner := 1;

<<inner\_loop>>

loop

v\_inner := v\_inner+1;

continue outer\_loop when v\_inner = 10;

dbms\_output.put\_line(' My inner value is : ' || v\_inner );

end loop inner\_loop;

end loop outer\_loop;

end;

* **GOTO STATEMENT**
* DECLARE

v\_searched\_number NUMBER := 22;

v\_is\_prime boolean := true;

BEGIN

FOR x in 2..v\_searched\_number-1 LOOP

IF v\_searched\_number MOD x = 0 THEN

dbms\_output.put\_line(v\_searched\_number|| ' is not a prime number..');

v\_is\_prime := false;

GOTO end\_point;

END IF;

END LOOP;

if v\_is\_prime then

dbms\_output.put\_line(v\_searched\_number|| ' is a prime number..');

end if;

<<end\_point>>

dbms\_output.put\_line('Check complete..');

END;

* DECLARE

v\_searched\_number NUMBER := 32457;

v\_is\_prime boolean := true;

x number := 2;

BEGIN

<<start\_point>>

IF v\_searched\_number MOD x = 0 THEN

dbms\_output.put\_line(v\_searched\_number|| ' is not a prime number..');

v\_is\_prime := false;

GOTO end\_point;

END IF;

x := x+1;

if x = v\_searched\_number then

goto prime\_point;

end if;

goto start\_point;

<<prime\_point>>

if v\_is\_prime then

dbms\_output.put\_line(v\_searched\_number|| ' is a prime number..');

end if;

<<end\_point>>

dbms\_output.put\_line('Check complete..');

END;

* **USE SQL IN PL/SQL**
* **SYNTAX**:

SELECT COLUMN/EXPRESSIONS **INTO** VARIABLES/RECORDS FROM TABLENAME

This can only fetch one row if we use into clause, otherwise it will give error

* declare

v\_name varchar2(50);

v\_salary employees.salary%type;

begin

select first\_name ||' '|| last\_name, salary into v\_name, v\_salary from employees where employee\_id = 130;

dbms\_output.put\_line('The salary of '|| v\_name || ' is : '|| v\_salary);

end;

* declare

v\_name varchar2(50);

sysdates employees.hire\_date%type;

begin

select first\_name ||' '|| last\_name, sysdates into v\_name, sysdates from employees where employee\_id = 130;

dbms\_output.put\_line('The salary of '|| v\_name || ' is : '|| sysdates);

end;

* declare

v\_name varchar2(50);

v\_sysdate employees.hire\_date%type;

employee\_id employees.employee\_id%type := 130;

begin

select first\_name ||' '|| last\_name, sysdate into v\_name, v\_sysdate from employees where employee\_id = employee\_id;

dbms\_output.put\_line('The salary of '|| v\_name || ' is : '|| v\_sysdate );

end;

* declare

v\_name varchar2(50);

v\_salary employees.salary%type;

v\_employee\_id employees.employee\_id%type := 130;

begin

select first\_name ||' '|| last\_name, salary into v\_name, v\_salary from employees where employee\_id = v\_employee\_id;

dbms\_output.put\_line('The salary of '|| v\_name || ' is : '|| v\_salary );

end;

* **DML OPERATIONS WITH PL/SQL**
* create table employees\_copy as select \* from employees;

DECLARE

v\_employee\_id pls\_integer := 0;

v\_salary\_increase number := 400;

begin

for i in 217..226 loop

insert into employees\_copy

(employee\_id,first\_name,last\_name,email,hire\_date,job\_id,salary)

values

(i, 'employee#'||i,'temp\_emp','abc@xmail.com',sysdate,'IT\_PROG',1000);

end loop;

end;

* create table employees\_copy as select \* from employees;

DECLARE

v\_employee\_id pls\_integer := 0;

v\_salary\_increase number := 400;

begin

for i in 217..226 loop

update employees\_copy

set salary = salary + v\_salary\_increase

where employee\_id = i;

end loop;

end;

* create table employees\_copy as select \* from employees;

DECLARE

v\_employee\_id pls\_integer := 0;

v\_salary\_increase number := 400;

begin

for i in 217..226 loop

delete from employees\_copy

where employee\_id = i;

end loop;

end;

* **Using Sequences in PL/SQL**
* Sequence is used where we ant to use incremental values, there we can use sequence.
* create sequence employee\_id\_seq

start with 207

increment by 1;

* begin

for i in 1..10 loop

insert into employees\_copy

(employee\_id,first\_name,last\_name,email,hire\_date,job\_id,salary)

values

(employee\_id\_seq.nextval, 'employee#'||**employee\_id\_seq.nextval**,'temp\_emp','abc@xmail.com',sysdate,'IT\_PROG',1000);

end loop;

end;

* declare

v\_seq\_num number;

begin

select employee\_id\_seq.nextval into v\_seq\_num from dual;

dbms\_output.put\_line(v\_seq\_num);

end;

* declare

v\_seq\_num number;

begin

select employee\_id\_seq.nextval into v\_seq\_num from employees\_copy where rownum = 1;

dbms\_output.put\_line(v\_seq\_num);

end;

* declare

v\_seq\_num number;

begin

v\_seq\_num := employee\_id\_seq.nextval;

dbms\_output.put\_line(v\_seq\_num);

end;

* begin

dbms\_output.put\_line(employee\_id\_seq.nextval);

end;

* begin

dbms\_output.put\_line(employee\_id\_seq.currval);

end;

* **PL/SQL RECORDS**
* declare

r\_emp employees%rowtype;

begin

select \* into r\_emp from employees where employee\_id = '101';

--r\_emp.salary := 2000;

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name|| ' earns '||r\_emp.salary||

' and hired at :' || r\_emp.hire\_date);

end;

* declare

--r\_emp employees%rowtype;

type t\_emp is record (first\_name varchar2(50),

last\_name employees.last\_name%type,

salary employees.salary%type,

hire\_date date);

r\_emp t\_emp;

begin

select first\_name,last\_name,salary,hire\_date into r\_emp

from employees where employee\_id = '101';

/\* r\_emp.first\_name := 'Alex';

r\_emp.salary := 2000;

r\_emp.hire\_date := '01-JAN-20'; \*/

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name|| ' earns '||r\_emp.salary||

' and hired at :' || r\_emp.hire\_date);

end;

* declare

type t\_edu is record (primary\_school varchar2(100),

high\_school varchar2(100),

university varchar2(100),

uni\_graduate\_date date

);

type t\_emp is record (first\_name varchar2(50),

last\_name employees.last\_name%type,

salary employees.salary%type NOT NULL DEFAULT 1000,

hire\_date date,

dept\_id employees.department\_id%type,

department departments%rowtype,

education t\_edu

);

r\_emp t\_emp;

begin

select first\_name,last\_name,salary,hire\_date,department\_id

into r\_emp.first\_name,r\_emp.last\_name,r\_emp.salary,r\_emp.hire\_date,r\_emp.dept\_id

from employees where employee\_id = '146';

select \* into r\_emp.department from departments where department\_id = r\_emp.dept\_id;

r\_emp.education.high\_school := 'Beverly Hills';

r\_emp.education.university := 'Oxford';

r\_emp.education.uni\_graduate\_date := '01-JAN-13';

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name|| ' earns '||r\_emp.salary||

' and hired at :' || r\_emp.hire\_date);

dbms\_output.put\_line('She graduated from '|| r\_emp.education.university|| ' at '|| r\_emp.education.uni\_graduate\_date);

dbms\_output.put\_line('Her Department Name is : '|| r\_emp.department.department\_name);

end;

* **EASY DML WITH RECORDS**
* create table retired\_employees as select \* from employees where 1=2;
* select \* from retired\_employees;
* declare

r\_emp employees%rowtype;

begin

select \* into r\_emp from employees where employee\_id = 104;

r\_emp.salary := 0;

r\_emp.commission\_pct := 0;

insert into retired\_employees values r\_emp;

end;

* declare

r\_emp employees%rowtype;

begin

select \* into r\_emp from employees where employee\_id = 104;

r\_emp.salary := 10;

r\_emp.commission\_pct := 0;

--insert into retired\_employees values r\_emp;

update retired\_employees set row = r\_emp where employee\_id = 104;

end;

* delete from retired\_employees;
* **COLLECTIONS**
* **NESTED TABLES—**is a key-value pair and also it is unbounded so we can have as many rows as we want, such that the key is increased by one
  + Key value pairs
* We can delete any values
* Not stored consecutively
* Nested tables are unbounded
* **VARRAYS—**
  + they are bounded in size at the time of declaration.
  + Its index starts from 1
  + Are one-dimensional
  + Are null by default
* **Associative Arrays—**this is very useful as keys can be strings and they ae also unbounded.
  + The key can have a string
  + Keys does not need to be sequential
  + Can have a scalar and record types
  + Do not initialize associative arrays
  + Associative arrays are indexed
* **VARRAYS**
* Declare

type e\_list is varray(5) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob','Richard');

for i in 1..5 loop

dbms\_output.put\_line(employees(i));

end loop;

end;

* **limit exceeding error example**

declare

type e\_list is varray(4) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob','Richard');

for i in 1..5 loop

dbms\_output.put\_line(employees(i));

end loop;

end;

* **Subscript beyond Bound error example**

declare

type e\_list is varray(5) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob');

for i in 1..5 loop

dbms\_output.put\_line(employees(i));

end loop;

end;

* **A working count() example**

declare

type e\_list is varray(5) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob');

for i in 1..employees.count() loop

dbms\_output.put\_line(employees(i));

end loop;

end;

* **A working first() last() functions example**

declare

type e\_list is varray(5) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob');

for i in employees.first()..employees.last() loop

dbms\_output.put\_line(employees(i));

end loop;

end;

* **A working exists() function example**

declare

type e\_list is varray(5) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob');

for i in 1..5 loop

if employees.exists(i) then

dbms\_output.put\_line(employees(i));

end if;

end loop;

end;

* **A working limit() function example**

declare

type e\_list is varray(5) of varchar2(50);

employees e\_list;

begin

employees := e\_list('Alex','Bruce','John','Bob');

dbms\_output.put\_line(employees.limit());

end;

* **A create-declare at the same time error example**

declare

type e\_list is varray(5) of varchar2(50);

employees e\_list('Alex','Bruce','John','Bob');

begin

-- employees := e\_list('Alex','Bruce','John','Bob');

for i in 1..5 loop

if employees.exists(i) then

dbms\_output.put\_line(employees(i));

end if;

end loop;

end;

* **A post insert varray example**

declare

type e\_list is varray(15) of varchar2(50);

employees e\_list := e\_list();

idx number := 1;

begin

for i in 100..110 loop

employees.extend;

select first\_name into employees(idx) from employees where employee\_id = i;

idx := idx + 1;

end loop;

for x in 1..employees.count() loop

dbms\_output.put\_line(employees(x));

end loop;

end;

* **An example for the schema level varray types**

create type e\_list is varray(15) of varchar2(50);

/

create or replace type e\_list as varray(20) of varchar2(100);

/

declare

employees e\_list := e\_list();

idx number := 1;

begin

for i in 100..110 loop

employees.extend;

select first\_name into employees(idx) from employees where employee\_id = i;

idx := idx + 1;

end loop;

for x in 1..employees.count() loop

dbms\_output.put\_line(employees(x));

end loop;

end;

/

DROP TYPE E\_LIST;

* **NESTED TABLES**
* **The simple usage of nested tables**

declare

type e\_list is table of varchar2(50);

emps e\_list;

begin

emps := e\_list('Alex','Bruce','John');

for i in 1..emps.count() loop

dbms\_output.put\_line(emps(i));

end loop;

end;

* **Adding a new value to a nested table after the initialization**

declare

type e\_list is table of varchar2(50);

emps e\_list;

begin

emps := e\_list('Alex','Bruce','John');

emps.extend;

emps(4) := 'Bob';

for i in 1..emps.count() loop

dbms\_output.put\_line(emps(i));

end loop;

end;

* **Adding values from the table declare**

type e\_list is table of employees.first\_name%type;

emps e\_list := e\_list();

idx pls\_integer := 1;

begin

for x in 100 .. 110 loop

emps.extend;

select first\_name into emps(idx) from employees where employee\_id = x;

idx := idx + 1;

end loop;

for i in 1..emps.count() loop

dbms\_output.put\_line(emps(i));

end loop;

end;

* delete example

declare

type e\_list is table of employees.first\_name%type;

emps e\_list := e\_list();

idx pls\_integer := 1;

begin

for x in 100 .. 110 loop

emps.extend;

select first\_name into emps(idx) from employees where employee\_id = x;

idx := idx + 1;

end loop;

emps.delete(3);

for i in 1..emps.count() loop

if emps.exists(i) then

dbms\_output.put\_line(emps(i));

end if;

end loop;

end;

* **ASSOCIATIVE ARRAYS**
* The first example

declare

type e\_list is table of employees.first\_name%type index by pls\_integer;

emps e\_list;

begin

for x in 100 .. 110 loop

select first\_name into emps(x) from employees

where employee\_id = x ;

end loop;

for i in emps.first()..emps.last() loop

dbms\_output.put\_line(emps(i));

end loop;

end;

* **Error example for the select into clause**

declare

type e\_list is table of employees.first\_name%type index by pls\_integer;

emps e\_list;

begin

for x in 100 .. 110 loop

select first\_name into emps(x) from employees

where employee\_id = x and department\_id = 60;

end loop;

for i in emps.first()..emps.last() loop

dbms\_output.put\_line(i);

end loop;

end;

* **Error example about reaching the empty indexdeclare**

type e\_list is table of employees.first\_name%type index by pls\_integer;

emps e\_list;

begin

emps(100) := 'Bob';

emps(120) := 'Sue';

for i in emps.first()..emps.last() loop

dbms\_output.put\_line(emps(i));

end loop;

end;

* **An example of iterating in associative arrays with while loops**

declare

type e\_list is table of employees.first\_name%type index by pls\_integer;

emps e\_list;

idx pls\_integer;

begin

emps(100) := 'Bob';

emps(120) := 'Sue';

idx := emps.first;

while idx is not null loop

dbms\_output.put\_line(emps(idx));

idx := emps.next(idx);

end loop;

end;

* **An example of using string based indexes with associative arrays**

declare

type e\_list is table of employees.first\_name%type index by employees.email%type;

emps e\_list;

idx employees.email%type;

v\_email employees.email%type;

v\_first\_name employees.first\_name%type;

begin

for x in 100 .. 110 loop

select first\_name,email into v\_first\_name,v\_email from employees

where employee\_id = x;

emps(v\_email) := v\_first\_name;

end loop;

idx := emps.first;

while idx is not null loop

dbms\_output.put\_line('The email of '|| emps(idx) ||' is : '|| idx);

idx := emps.next(idx);

end loop;

end;

* **An example of using associative arrays with records**

--------------- declare

type e\_list is table of employees%rowtype index by employees.email%type;

emps e\_list;

idx employees.email%type;

begin

for x in 100 .. 110 loop

select \* into emps(x) from employees

where employee\_id = x;

end loop;

idx := emps.first;

while idx is not null loop

dbms\_output.put\_line('The email of '|| emps(idx).first\_name

||' '||emps(idx).last\_name||' is : '|| emps(idx).email);

idx := emps.next(idx);

end loop;

end;

* **An example of using associative arrays with record types**

declare

type e\_type is record (first\_name employees.first\_name%type,

last\_name employees.last\_name%type,

email employees.email%type);

type e\_list is table of e\_type index by employees.email%type;

emps e\_list;

idx employees.email%type;

begin

for x in 100 .. 110 loop

select first\_name,last\_name,email into emps(x) from employees

where employee\_id = x;

end loop;

idx := emps.first;

while idx is not null loop

dbms\_output.put\_line('The email of '|| emps(idx).first\_name

||' '||emps(idx).last\_name||' is : '|| emps(idx).email);

idx := emps.next(idx);

end loop;

end;

* **An example of printing from the last to the first**

declare

type e\_type is record (first\_name employees.first\_name%type,

last\_name employees.last\_name%type,

email employees.email%type);

type e\_list is table of e\_type index by employees.email%type;

emps e\_list;

idx employees.email%type;

begin

for x in 100 .. 110 loop

select first\_name,last\_name,email into emps(x) from employees

where employee\_id = x;

end loop;

--emps.delete(100,104);

idx := emps.last;

while idx is not null loop

dbms\_output.put\_line('The email of '|| emps(idx).first\_name

||' '||emps(idx).last\_name||' is : '|| emps(idx).email);

idx := emps.prior(idx);

end loop;

end;

* **An example of inserting with associative arrays**

create table employees\_salary\_history as select \* from employees where 1=2;

alter table employees\_salary\_history add insert\_date date;

select \* from employees\_salary\_history;

/

declare

type e\_list is table of employees\_salary\_history%rowtype index by pls\_integer;

emps e\_list;

idx pls\_integer;

begin

for x in 100 .. 110 loop

select e.\*,'01-JUN-20' into emps(x) from employees e

where employee\_id = x;

end loop;

idx := emps.first;

while idx is not null loop

emps(idx).salary := emps(idx).salary + emps(idx).salary\*0.2;

insert into employees\_salary\_history values emps(idx);

dbms\_output.put\_line('The employee '|| emps(idx).first\_name

||' is inserted to the history table');

idx := emps.next(idx);

end loop;

end;

/

drop table employees\_salary\_history;

* **STORING COLLECTIONS IN TABLES**
  + **Storing Varray Example**

create or replace type t\_phone\_number as object (p\_type varchar2(10), p\_number varchar2(50));

/

create or replace type v\_phone\_numbers as varray(3) of t\_phone\_number;

/

create table emps\_with\_phones (employee\_id number,

first\_name varchar2(50),

last\_name varchar2(50),

phone\_number v\_phone\_numbers);

/

select \* from emps\_with\_phones;

/

insert into emps\_with\_phones values (10,'Alex','Brown',v\_phone\_numbers(

t\_phone\_number('HOME','111.111.1111'),

t\_phone\_number('WORK','222.222.2222'),

t\_phone\_number('MOBILE','333.333.3333')

));

insert into emps\_with\_phones values (11,'Bob','Green',v\_phone\_numbers(

t\_phone\_number('HOME','000.000.000'),

t\_phone\_number('WORK','444.444.4444')

));

/

---------------Querying the varray example

select e.first\_name,last\_name,p.p\_type,p.p\_number from emps\_with\_phones e, table(e.phone\_number) p;

* + **The codes for the storing nested table example**

create or replace type n\_phone\_numbers as table of t\_phone\_number;

/

create table emps\_with\_phones2 (employee\_id number,

first\_name varchar2(50),

last\_name varchar2(50),

phone\_number n\_phone\_numbers)

NESTED TABLE phone\_number STORE AS phone\_numbers\_table;

/

select \* from emps\_with\_phones2;

/

insert into emps\_with\_phones2 values (10,'Alex','Brown',n\_phone\_numbers(

t\_phone\_number('HOME','111.111.1111'),

t\_phone\_number('WORK','222.222.2222'),

t\_phone\_number('MOBILE','333.333.3333')

));

insert into emps\_with\_phones2 values (11,'Bob','Green',n\_phone\_numbers(

t\_phone\_number('HOME','000.000.000'),

t\_phone\_number('WORK','444.444.4444')

));

/

select e.first\_name,last\_name,p.p\_type,p.p\_number from emps\_with\_phones2 e, table(e.phone\_number) p;

* + **new insert and update**

insert into emps\_with\_phones2 values (11,'Bob','Green',n\_phone\_numbers(

t\_phone\_number('HOME','000.000.000'),

t\_phone\_number('WORK','444.444.4444'),

t\_phone\_number('WORK2','444.444.4444'),

t\_phone\_number('WORK3','444.444.4444'),

t\_phone\_number('WORK4','444.444.4444'),

t\_phone\_number('WORK5','444.444.4444')

));

select \* from emps\_with\_phones2;

update emps\_with\_phones2 set phone\_number = n\_phone\_numbers(

t\_phone\_number('HOME','000.000.000'),

t\_phone\_number('WORK','444.444.4444'),

t\_phone\_number('WORK2','444.444.4444'),

t\_phone\_number('WORK3','444.444.4444'),

t\_phone\_number('WORK4','444.444.4444'),

t\_phone\_number('WORK5','444.444.4444')

)

where employee\_id = 11;

* + **Adding a new value into the nested table inside of a table**

declare

p\_num n\_phone\_numbers;

begin

select phone\_number into p\_num from emps\_with\_phones2 where employee\_id = 10;

p\_num.extend;

p\_num(5) := t\_phone\_number('FAX','999.99.9999');

UPDATE emps\_with\_phones2 set phone\_number = p\_num where employee\_id = 10;

end;

* **HANDLING THE EXCEPTION**
  + declare

v\_name varchar2(6);

begin

select first\_name into v\_name from employees where employee\_id = 50;

dbms\_output.put\_line('Hello');

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the selected id');

end;

* + **Handling multiple exceptions**

declare

v\_name varchar2(6);

v\_department\_name varchar2(100);

begin

select first\_name into v\_name from employees where employee\_id = 100;

select department\_id into v\_department\_name from employees where first\_name = v\_name;

dbms\_output.put\_line('Hello '|| v\_name || '. Your department id is : '|| v\_department\_name );

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the selected id');

when too\_many\_rows then

dbms\_output.put\_line('There are more than one employees with the name '|| v\_name);

dbms\_output.put\_line('Try with a different employee');

end;

* + **when others then example**

declare

v\_name varchar2(6);

v\_department\_name varchar2(100);

begin

select first\_name into v\_name from employees where employee\_id = 103;

select department\_id into v\_department\_name from employees where first\_name = v\_name;

dbms\_output.put\_line('Hello '|| v\_name || '. Your department id is : '|| v\_department\_name );

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the selected id');

when too\_many\_rows then

dbms\_output.put\_line('There are more than one employees with the name '|| v\_name);

dbms\_output.put\_line('Try with a different employee');

when others then

dbms\_output.put\_line('An unexpected error happened. Connect with the programmer..');

end;

* + **sqlerrm & sqlcode example**

declare

v\_name varchar2(6);

v\_department\_name varchar2(100);

begin

select first\_name into v\_name from employees where employee\_id = 103;

select department\_id into v\_department\_name from employees where first\_name = v\_name;

dbms\_output.put\_line('Hello '|| v\_name || '. Your department id is : '|| v\_department\_name );

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the selected id');

when too\_many\_rows then

dbms\_output.put\_line('There are more than one employees with the name '|| v\_name);

dbms\_output.put\_line('Try with a different employee');

when others then

dbms\_output.put\_line('An unexpected error happened. Connect with the programmer..');

dbms\_output.put\_line(sqlcode || ' ---> '|| sqlerrm);

end;

* + **Inner block exception example**

declare

v\_name varchar2(6);

v\_department\_name varchar2(100);

begin

select first\_name into v\_name from employees where employee\_id = 100;

begin

select department\_id into v\_department\_name from employees where first\_name = v\_name;

exception

when too\_many\_rows then

v\_department\_name := 'Error in department\_name';

end;

dbms\_output.put\_line('Hello '|| v\_name || '. Your department id is : '|| v\_department\_name );

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the selected id');

when too\_many\_rows then

dbms\_output.put\_line('There are more than one employees with the name '|| v\_name);

dbms\_output.put\_line('Try with a different employee');

when others then

dbms\_output.put\_line('An unexpected error happened. Connect with the programmer..');

dbms\_output.put\_line(sqlcode || ' ---> '|| sqlerrm);

end;

/

select \* from employees where first\_name = 'Steven';

* **NON PREDEFINED EXCEPTION**
  + **Is an unnamed exception and cannot trap with error codes.**
  + **But we can define a exception by using**
    - **Declare exceptionName exception;**
    - **PRAGMA exception\_init(exceptionName,error\_code)**
* begin

UPDATE employees\_copy set email = null where employee\_id = 100;

end;

-----------------HANDLING a nonpredefined exception

declare

cannot\_update\_to\_null exception;

pragma exception\_init(cannot\_update\_to\_null,-01407);

begin

UPDATE employees\_copy set email = null where employee\_id = 100;

exception

when cannot\_update\_to\_null then

dbms\_output.put\_line('You cannot update with a null value!');

end;

* **USER DEFINED EXCEPTION**
* **We need to handle some exceptions about ur business**
* **These exceptions are not error for the database**
* **Define user-defined exception and raise it**
* **Exception\_name exception;**
* **RAISE Exception\_name;**
* declare

too\_high\_salary exception;

v\_salary\_check pls\_integer;

begin

select salary into v\_salary\_check from employees where employee\_id = 100;

if v\_salary\_check > 20000 then

raise too\_high\_salary;

end if;

--we do our business if the salary is under 2000

dbms\_output.put\_line('The salary is in an acceptable range');

exception

when too\_high\_salary then

dbms\_output.put\_line('This salary is too high. You need to decrease it.');

end;

* **raising a predefined exception**

declare

too\_high\_salary exception;

v\_salary\_check pls\_integer;

begin

select salary into v\_salary\_check from employees where employee\_id = 100;

if v\_salary\_check > 20000 then

raise invalid\_number;

end if;

--we do our business if the salary is under 2000

dbms\_output.put\_line('The salary is in an acceptable range');

exception

when invalid\_number then

dbms\_output.put\_line('This salary is too high. You need to decrease it.');

end;

* **raising inside of the exception**

declare

too\_high\_salary exception;

v\_salary\_check pls\_integer;

begin

select salary into v\_salary\_check from employees where employee\_id = 100;

if v\_salary\_check > 20000 then

raise invalid\_number;

end if;

--we do our business if the salary is under 2000

dbms\_output.put\_line('The salary is in an acceptable range');

exception

when invalid\_number then

dbms\_output.put\_line('This salary is too high. You need to decrease it.');

raise;

end;

* **CURSORS**
  + **Cursors are pointers to the data**
  + **There are 2 types of cursors**
    - **Implicit Cursor**
    - **Explicit Cursor**
* **We can control the cursors**
* **Explicit cursors are controlled by programmer**
* **Collections vs Cursors**
* **You cannot go back in cursors**
* **Cursors are the pointers that you can iterate through the data that you selected**
* **EXPLICIT CURSORS**
* **USAGE**
  + **Declare—is a result\_set of select query**

**Cursor cursor\_name is select statement**

**Inside Begin/end Block**

* + **Open**

**Open cursor\_name**

* + **Fetch—when the cursor fetches a row, the pointer goes to the next row of active set.**

**Fetch cursor\_name into variables,records;**

* + **Check**

**Optional to use**

* + **Close**

**Close cursor\_name**

* declare

cursor c\_emps is select first\_name,last\_name from employees;

v\_first\_name employees.first\_name%type;

v\_last\_name employees.last\_name%type;

begin

open c\_emps;

fetch c\_emps into v\_first\_name,v\_last\_name;

fetch c\_emps into v\_first\_name,v\_last\_name;

fetch c\_emps into v\_first\_name,v\_last\_name;

dbms\_output.put\_line(v\_first\_name|| ' ' || v\_last\_name);

fetch c\_emps into v\_first\_name,v\_last\_name;

dbms\_output.put\_line(v\_first\_name|| ' ' || v\_last\_name);

close c\_emps;

end;

* **cursor with join example**

declare

cursor c\_emps is select first\_name,last\_name, department\_name from employees

join departments using (department\_id)

where department\_id between 30 and 60;

v\_first\_name employees.first\_name%type;

v\_last\_name employees.last\_name%type;

v\_department\_name departments.department\_name%type;

begin

open c\_emps;

fetch c\_emps into v\_first\_name, v\_last\_name,v\_department\_name;

dbms\_output.put\_line(v\_first\_name|| ' ' || v\_last\_name|| ' in the department of '|| v\_department\_name);

close c\_emps;

end;

* **Cursors with Records**
* declare

type r\_emp is record ( v\_first\_name employees.first\_name%type,

v\_last\_name employees.last\_name%type);

v\_emp r\_emp;

cursor c\_emps is select first\_name,last\_name from employees;

begin

open c\_emps;

fetch c\_emps into v\_emp;

dbms\_output.put\_line(v\_emp.v\_first\_name|| ' ' || v\_emp.v\_last\_name);

close c\_emps;

end;

* **An example for using cursors table rowtype**

declare

v\_emp employees%rowtype;

cursor c\_emps is select first\_name,last\_name from employees;

begin

open c\_emps;

fetch c\_emps into v\_emp.first\_name,v\_emp.last\_name;

dbms\_output.put\_line(v\_emp.first\_name|| ' ' || v\_emp.last\_name);

close c\_emps;

end;

* **An example for using cursors with cursor%rowtype.**

declare

cursor c\_emps is select first\_name,last\_name from employees;

v\_emp c\_emps%rowtype;

begin

open c\_emps;

fetch c\_emps into v\_emp.first\_name,v\_emp.last\_name;

dbms\_output.put\_line(v\_emp.first\_name|| ' ' || v\_emp.last\_name);

close c\_emps;

end;

* **LOOPING WITH CURSORS**
* **%NOTFOUND Function**—is used if there is no new row found
* **%FOUND**-- function is used if there is no next row found
* declare

cursor c\_emps is select \* from employees where department\_id = 30;

v\_emps c\_emps%rowtype;

begin

open c\_emps;

loop

fetch c\_emps into v\_emps;

dbms\_output.put\_line(v\_emps.employee\_id|| ' ' ||v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **notfound example**

declare

cursor c\_emps is select \* from employees where department\_id = 30;

v\_emps c\_emps%rowtype;

begin

open c\_emps;

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound;

dbms\_output.put\_line(v\_emps.employee\_id|| ' ' ||v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **while loop example**

declare

cursor c\_emps is select \* from employees where department\_id = 30;

v\_emps c\_emps%rowtype;

begin

open c\_emps;

fetch c\_emps into v\_emps;

while c\_emps%found loop

dbms\_output.put\_line(v\_emps.employee\_id|| ' ' ||v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

fetch c\_emps into v\_emps;

--exit when c\_emps%notfound;

end loop;

close c\_emps;

end;

* **for loop with cursor example**

declare

cursor c\_emps is select \* from employees where department\_id = 30;

v\_emps c\_emps%rowtype;

begin

open c\_emps;

for i in 1..6 loop

fetch c\_emps into v\_emps;

dbms\_output.put\_line(v\_emps.employee\_id|| ' ' ||v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **FOR..IN clause example**

declare

cursor c\_emps is select \* from employees where department\_id = 30;

begin

for i in c\_emps loop

dbms\_output.put\_line(i.employee\_id|| ' ' ||i.first\_name|| ' ' ||i.last\_name);

end loop;

end;

* **FOR..IN with select example**

begin

for i in (select \* from employees where department\_id = 30) loop

dbms\_output.put\_line(i.employee\_id|| ' ' ||i.first\_name|| ' ' ||i.last\_name);

end loop;

end;

* **CURSORS WITH PARAMETERS**
* **SYNTAX:**

**Declare**

**Cursor cursor\_name(parameter\_name,datatype..) is select statement**

**Begin**

**Open cursor\_name(parameters);**

**Fetch cursor\_name into variables, records;**

**close cursor\_name;**

**end;**

* declare

cursor c\_emps (p\_dept\_id number) is select first\_name,last\_name,department\_name

from employees join departments using (department\_id)

where department\_id = p\_dept\_id;

v\_emps c\_emps%rowtype;

begin

open c\_emps(20);

fetch c\_emps into v\_emps;

dbms\_output.put\_line('The employees in department of '|| v\_emps.department\_name|| ' are :');

close c\_emps;

open c\_emps(20);

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound;

dbms\_output.put\_line(v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **bind variables as parameters**

declare

cursor c\_emps (p\_dept\_id number) is select first\_name,last\_name,department\_name

from employees join departments using (department\_id)

where department\_id = p\_dept\_id;

v\_emps c\_emps%rowtype;

begin

open c\_emps(:b\_emp);

fetch c\_emps into v\_emps;

dbms\_output.put\_line('The employees in department of '|| v\_emps.department\_name|| ' are :');

close c\_emps;

open c\_emps(:b\_emp);

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound;

dbms\_output.put\_line(v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **cursors with two different parameters**

declare

cursor c\_emps (p\_dept\_id number) is select first\_name,last\_name,department\_name

from employees join departments using (department\_id)

where department\_id = p\_dept\_id;

v\_emps c\_emps%rowtype;

begin

open c\_emps(:b\_dept\_id);

fetch c\_emps into v\_emps;

dbms\_output.put\_line('The employees in department of '|| v\_emps.department\_name|| ' are :');

close c\_emps;

open c\_emps(:b\_dept\_id);

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound;

dbms\_output.put\_line(v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

open c\_emps(:b\_dept\_id2);

fetch c\_emps into v\_emps;

dbms\_output.put\_line('The employees in department of '|| v\_emps.department\_name|| ' are :');

close c\_emps;

open c\_emps(:b\_dept\_id2);

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound;

dbms\_output.put\_line(v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **cursor with parameters - for in loops**

declare

cursor c\_emps (p\_dept\_id number) is select first\_name,last\_name,department\_name

from employees join departments using (department\_id)

where department\_id = p\_dept\_id;

v\_emps c\_emps%rowtype;

begin

open c\_emps(:b\_dept\_id);

fetch c\_emps into v\_emps;

dbms\_output.put\_line('The employees in department of '|| v\_emps.department\_name|| ' are :');

close c\_emps;

open c\_emps(:b\_dept\_id);

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound;

dbms\_output.put\_line(v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

open c\_emps(:b\_dept\_id2);

fetch c\_emps into v\_emps;

dbms\_output.put\_line('The employees in department of '|| v\_emps.department\_name|| ' are :');

close c\_emps;

for i in c\_emps(:b\_dept\_id2) loop

dbms\_output.put\_line(i.first\_name|| ' ' ||i.last\_name);

end loop;

end;

* **cursors with multiple parameters**

declare

cursor c\_emps (p\_dept\_id number , p\_job\_id varchar2) is select first\_name,last\_name,job\_id,department\_name

from employees join departments using (department\_id)

where department\_id = p\_dept\_id

and job\_id = p\_job\_id;

v\_emps c\_emps%rowtype;

begin

for i in c\_emps(50,'ST\_MAN') loop

dbms\_output.put\_line(i.first\_name|| ' ' ||i.last\_name|| ' - ' || i.job\_id);

end loop;

dbms\_output.put\_line(' - ');

for i in c\_emps(80,'SA\_MAN') loop

dbms\_output.put\_line(i.first\_name|| ' ' ||i.last\_name|| ' - ' || i.job\_id);

end loop;

end;

* **An error example of using parameter name with the column name**

declare

cursor c\_emps (p\_dept\_id number , job\_id varchar2) is select first\_name,last\_name,job\_id,department\_name

from employees join departments using (department\_id)

where department\_id = p\_dept\_id

and job\_id = job\_id;

v\_emps c\_emps%rowtype;

begin

for i in c\_emps(50,'ST\_MAN') loop

dbms\_output.put\_line(i.first\_name|| ' ' ||i.last\_name|| ' - ' || i.job\_id);

end loop;

dbms\_output.put\_line(' - ');

for i in c\_emps(80,'SA\_MAN') loop

dbms\_output.put\_line(i.first\_name|| ' ' ||i.last\_name|| ' - ' || i.job\_id);

end loop;

end;

* **CURSOR WITH ATTRIBUTES**
* **%FOUND—Returns true if fetch returned a row.**
* **%NOTFOUND—Opposite of found**
* **%ISOPEN—Returns true if cursor is opened**
* **%ROWCOUNT—Returns the number of fetched rows**
* declare

cursor c\_emps is select \* from employees where department\_id = 50;

v\_emps c\_emps%rowtype;

begin

if not c\_emps%isopen then

open c\_emps;

dbms\_output.put\_line('hello');

end if;

dbms\_output.put\_line(c\_emps%rowcount);

fetch c\_emps into v\_emps;

dbms\_output.put\_line(c\_emps%rowcount);

dbms\_output.put\_line(c\_emps%rowcount);

fetch c\_emps into v\_emps;

dbms\_output.put\_line(c\_emps%rowcount);

close c\_emps;

open c\_emps;

loop

fetch c\_emps into v\_emps;

exit when c\_emps%notfound or c\_emps%rowcount>5;

dbms\_output.put\_line(c\_emps%rowcount|| ' ' ||v\_emps.first\_name|| ' ' ||v\_emps.last\_name);

end loop;

close c\_emps;

end;

* **FOR UPDATE CLAUSE**
* **When we update a row, it is locked to the others**
* **“For update” clause locks the selected row until we commit**
* **Nowait options will terminate execution if there is a lock**
* **Default option is wait**
* **“’For update of” only locks the selected tables**
* **SYNTAX**

**Cursor cursor\_name (parameter\_name datatype..) is select statement**

**For update [of column(s)] [nowait | wait n]**

* grant create session to my\_user;

grant select any table to my\_user;

grant update on hr.employees\_copy to my\_user;

grant update on hr.departments to my\_user;

UPDATE EMPLOYEES\_COPY SET PHONE\_NUMBER = '1' WHERE EMPLOYEE\_ID = 100;

declare

cursor c\_emps is select employee\_id,first\_name,last\_name,department\_name

from employees\_copy join departments using (department\_id)

where employee\_id in (100,101,102)

for update;

begin

/\* for r\_emps in c\_emps loop

update employees\_copy set phone\_number = 3

where employee\_id = r\_emps.employee\_id;

end loop; \*/

open c\_emps;

end;

* **example of wait with second**

declare

cursor c\_emps is select employee\_id,first\_name,last\_name,department\_name

from employees\_copy join departments using (department\_id)

where employee\_id in (100,101,102)

for update of employees\_copy.phone\_number,

departments.location\_id wait 5;

begin

/\* for r\_emps in c\_emps loop

update employees\_copy set phone\_number = 3

where employee\_id = r\_emps.employee\_id;

end loop; \*/

open c\_emps;

end;

* **example of nowait**

declare

cursor c\_emps is select employee\_id,first\_name,last\_name,department\_name

from employees\_copy join departments using (department\_id)

where employee\_id in (100,101,102)

for update of employees\_copy.phone\_number,

departments.location\_id nowait;

begin

/\* for r\_emps in c\_emps loop

update employees\_copy set phone\_number = 3

where employee\_id = r\_emps.employee\_id;

end loop; \*/

open c\_emps;

end;

* **WHERE CURRENT OF CLAUSE**
* **We use this when we need to make some or deletes one by one**
* declare

cursor c\_emps is select \* from employees

where department\_id = 30 for update;

begin

for r\_emps in c\_emps loop

update employees set salary = salary + 60

where current of c\_emps;

end loop;

end;

* **Wrong example of using where current of clause**

declare

cursor c\_emps is select e.\* from employees e, departments d

where

e.department\_id = d.department\_id

and e.department\_id = 30 for update;

begin

for r\_emps in c\_emps loop

update employees set salary = salary + 60

where current of c\_emps;

end loop;

end;

* **An example of using rowid like where current of clause**

declare

cursor c\_emps is select e.rowid,e.salary from employees e, departments d

where

e.department\_id = d.department\_id

and e.department\_id = 30 for update;

begin

for r\_emps in c\_emps loop

update employees set salary = salary + 60

where rowid = r\_emps.rowid;

end loop;

end;

* **REFERENCE CURSORS**
* **Reference cursors are the pointers to the actual cursors**
* **You can use cursor with multiple queries**
* **We cannot:**
  + **Assign null values**
  + **Use in table-view create codes**
  + **Store in collections**
  + **Compare**
* **Twio types of Ref Cursors**
  + **Strong(restrictive cursors)**
  + **Weak(non restrictive)cursors**
* **SYNTAX**

**type cursor\_type\_name is ref cursor[return return type]**

**open cursor\_variable\_name for query;**

* declare

type t\_emps is ref cursor return employees%rowtype;

rc\_emps t\_emps;

r\_emps employees%rowtype;

begin

open rc\_emps for select \* from employees;

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name);

end loop;

close rc\_emps;

end;

* **in two different queries**

declare

type t\_emps is ref cursor return employees%rowtype;

rc\_emps t\_emps;

r\_emps employees%rowtype;

begin

open rc\_emps for select \* from retired\_employees;

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name);

end loop;

close rc\_emps;

dbms\_output.put\_line('--------------');

open rc\_emps for select \* from employees where job\_id = 'IT\_PROG';

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name);

end loop;

close rc\_emps;

end;

* **Example of using with %type when declaring records first**

declare

r\_emps employees%rowtype;

type t\_emps is ref cursor return r\_emps%type;

rc\_emps t\_emps;

--type t\_emps2 is ref cursor return rc\_emps%rowtype;

begin

open rc\_emps for select \* from retired\_employees;

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name);

end loop;

close rc\_emps;

dbms\_output.put\_line('--------------');

open rc\_emps for select \* from employees where job\_id = 'IT\_PROG';

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name);

end loop;

close rc\_emps;

end;

* **manually declared record type with cursors example**

declare

type ty\_emps is record (e\_id number,

first\_name employees.last\_name%type,

last\_name employees.last\_name%type,

department\_name departments.department\_name%type);

r\_emps ty\_emps;

type t\_emps is ref cursor return ty\_emps;

rc\_emps t\_emps;

begin

open rc\_emps for select employee\_id,first\_name,last\_name,department\_name

from employees join departments using (department\_id);

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name||

' is at the department of : '|| r\_emps.department\_name );

end loop;

close rc\_emps;

end;

---------------first example of weak ref cursors

declare

type ty\_emps is record (e\_id number,

first\_name employees.last\_name%type,

last\_name employees.last\_name%type,

department\_name departments.department\_name%type);

r\_emps ty\_emps;

type t\_emps is ref cursor;

rc\_emps t\_emps;

q varchar2(200);

begin

q := 'select employee\_id,first\_name,last\_name,department\_name

from employees join departments using (department\_id)';

open rc\_emps for q;

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name||

' is at the department of : '|| r\_emps.department\_name );

end loop;

close rc\_emps;

end;

* **bind variables with cursors example**

declare

type ty\_emps is record (e\_id number,

first\_name employees.last\_name%type,

last\_name employees.last\_name%type,

department\_name departments.department\_name%type);

r\_emps ty\_emps;

type t\_emps is ref cursor;

rc\_emps t\_emps;

r\_depts departments%rowtype;

--r t\_emps%rowtype;

q varchar2(200);

begin

q := 'select employee\_id,first\_name,last\_name,department\_name

from employees join departments using (department\_id)

where department\_id = :t';

open rc\_emps for q using '50';

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name||

' is at the department of : '|| r\_emps.department\_name );

end loop;

close rc\_emps;

open rc\_emps for select \* from departments;

loop

fetch rc\_emps into r\_depts;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_depts.department\_id|| ' ' || r\_depts.department\_name);

end loop;

close rc\_emps;

end;

* **sys\_refcursor example**

declare

type ty\_emps is record (e\_id number,

first\_name employees.last\_name%type,

last\_name employees.last\_name%type,

department\_name departments.department\_name%type);

r\_emps ty\_emps;

-- type t\_emps is ref cursor;

rc\_emps sys\_refcursor;

r\_depts departments%rowtype;

--r t\_emps%rowtype;

q varchar2(200);

begin

q := 'select employee\_id,first\_name,last\_name,department\_name

from employees join departments using (department\_id)

where department\_id = :t';

open rc\_emps for q using '50';

loop

fetch rc\_emps into r\_emps;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_emps.first\_name|| ' ' || r\_emps.last\_name||

' is at the department of : '|| r\_emps.department\_name );

end loop;

close rc\_emps;

open rc\_emps for select \* from departments;

loop

fetch rc\_emps into r\_depts;

exit when rc\_emps%notfound;

dbms\_output.put\_line(r\_depts.department\_id|| ' ' || r\_depts.department\_name);

end loop;

close rc\_emps;

end;

* **STORED PROCEDURES:**
* **Reusability is important in programming**
* **What are the differences than anonymous blocks:**
  + **Stored in database with names**
  + **Compiled only once**
  + **Can be called by another block or application**
  + **Can return values**
  + **Can take parameters**
* **Creating and Using Procedures(SYNTAX)**

**CREATE [or REPLACE] procedure procedure\_name[parameter\_name [IN|OUT] type[,…..])] {IS | AS}**

**BEGIN**

**---------------**

**EXCEPTION**

**---------------**

**END;**

* **Creating a procedure**

create procedure increase\_salaries as

cursor c\_emps is select \* from employees\_copy for update;

v\_salary\_increase number := 1.10;

v\_old\_salary number;

begin

for r\_emp in c\_emps loop

v\_old\_salary := r\_emp.salary;

r\_emp.salary := r\_emp.salary \* v\_salary\_increase + r\_emp.salary \* nvl(r\_emp.commission\_pct,0);

update employees\_copy set row = r\_emp where current of c\_emps;

dbms\_output.put\_line('The salary of : '|| r\_emp.employee\_id

|| ' is increased from '||v\_old\_salary||' to '||r\_emp.salary);

end loop;

end;

* **Multiple procedure usage**

begin

dbms\_output.put\_line('Increasing the salaries!...');

INCREASE\_SALARIES;

INCREASE\_SALARIES;

INCREASE\_SALARIES;

INCREASE\_SALARIES;

dbms\_output.put\_line('All the salaries are successfully increased!...');

end;

* **Different procedures in one block**

begin

dbms\_output.put\_line('Increasing the salaries!...');

INCREASE\_SALARIES;

new\_line;

INCREASE\_SALARIES;

new\_line;

INCREASE\_SALARIES;

new\_line;

INCREASE\_SALARIES;

dbms\_output.put\_line('All the salaries are successfully increased!...');

end;

* **Creating a procedure to ease the dbms\_output.put\_line procedure**

create procedure new\_line as

begin

dbms\_output.put\_line('------------------------------------------');

end;

* **Modifying the procedure with using the OR REPLACE command.**

create or replace procedure increase\_salaries as

cursor c\_emps is select \* from employees\_copy for update;

v\_salary\_increase number := 1.10;

v\_old\_salary number;

begin

for r\_emp in c\_emps loop

v\_old\_salary := r\_emp.salary;

r\_emp.salary := r\_emp.salary \* v\_salary\_increase + r\_emp.salary \* nvl(r\_emp.commission\_pct,0);

update employees\_copy set row = r\_emp where current of c\_emps;

dbms\_output.put\_line('The salary of : '|| r\_emp.employee\_id

|| ' is increased from '||v\_old\_salary||' to '||r\_emp.salary);

end loop;

dbms\_output.put\_line('Procedure finished executing!');

end

* **CREATING PROCEDURE WITH IN/OUT PARAMETERS**
  + **IN- is used for passing in the parameters to function/procedure**
  + **OUT- is used for returning the parameters from function/procedure**
* **Creating a procedure with the IN parameters**

create or replace procedure increase\_salaries (v\_salary\_increase in number, v\_department\_id pls\_integer) as

cursor c\_emps is select \* from employees\_copy where department\_id = v\_department\_id for update;

v\_old\_salary number;

begin

for r\_emp in c\_emps loop

v\_old\_salary := r\_emp.salary;

r\_emp.salary := r\_emp.salary \* v\_salary\_increase + r\_emp.salary \* nvl(r\_emp.commission\_pct,0);

update employees\_copy set row = r\_emp where current of c\_emps;

dbms\_output.put\_line('The salary of : '|| r\_emp.employee\_id

|| ' is increased from '||v\_old\_salary||' to '||r\_emp.salary);

end loop;

dbms\_output.put\_line('Procedure finished executing!');

end;

* **Creating a procedure with the OUT parameters**

create or replace procedure increase\_salaries

(v\_salary\_increase in out number, v\_department\_id pls\_integer, v\_affected\_employee\_count out number) as

cursor c\_emps is select \* from employees\_copy where department\_id = v\_department\_id for update;

v\_old\_salary number;

v\_sal\_inc number := 0;

begin

v\_affected\_employee\_count := 0;

for r\_emp in c\_emps loop

v\_old\_salary := r\_emp.salary;

r\_emp.salary := r\_emp.salary \* v\_salary\_increase + r\_emp.salary \* nvl(r\_emp.commission\_pct,0);

update employees\_copy set row = r\_emp where current of c\_emps;

dbms\_output.put\_line('The salary of : '|| r\_emp.employee\_id

|| ' is increased from '||v\_old\_salary||' to '||r\_emp.salary);

v\_affected\_employee\_count := v\_affected\_employee\_count + 1;

v\_sal\_inc := v\_sal\_inc + v\_salary\_increase + nvl(r\_emp.commission\_pct,0);

end loop;

v\_salary\_increase := v\_sal\_inc / v\_affected\_employee\_count;

dbms\_output.put\_line('Procedure finished executing!');

end;

* **Another example of creating a procedure with the IN parameter**

CREATE OR REPLACE PROCEDURE PRINT(TEXT IN VARCHAR2) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(TEXT);

END;

-----------------Using the procedures that has the IN parameters

begin

PRINT('SALARY INCREASE STARTED!..');

INCREASE\_SALARIES(1.15,90);

PRINT('SALARY INCREASE FINISHED!..');

end;

-----------------Using the procedure that has OUT parameters

declare

v\_sal\_inc number := 1.2;

v\_aff\_emp\_count number;

begin

PRINT('SALARY INCREASE STARTED!..');

INCREASE\_SALARIES(v\_sal\_inc,80,v\_aff\_emp\_count);

PRINT('The affected employee count is : '|| v\_aff\_emp\_count);

PRINT('The average salary increase is : '|| v\_sal\_inc || ' percent!..');

PRINT('SALARY INCREASE FINISHED!..');

end;

* **Named & Mixed Notations and Default Option**
* **A standard procedure creation with a default value**

create or replace PROCEDURE PRINT(TEXT IN VARCHAR2 := 'This is the print function!.') IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(TEXT);

END;

* **Executing a procedure without any parameter. It runs because it has a default value.**

exec print();

* **Running a procedure with null value will not use the default value**

exec print(null);

* **Procedure creation of a default value usage**

create or replace procedure add\_job(job\_id pls\_integer, job\_title varchar2,

min\_salary number default 1000, max\_salary number default null) is

begin

insert into jobs values (job\_id,job\_title,min\_salary,max\_salary);

print('The job : '|| job\_title || ' is inserted!..');

end;

* **A standard run of the procedure**

exec ADD\_JOB('IT\_DIR','IT Director',5000,20000);

* **Running a procedure with using the default values**

exec ADD\_JOB('IT\_DIR2','IT Director',5000);

* **Running a procedure with the named notation**

exec ADD\_JOB('IT\_DIR5','IT Director',max\_salary=>10000);

* **Running a procedure with the named notation example 2**

exec ADD\_JOB(job\_title=>'IT Director',job\_id=>'IT\_DIR7',max\_salary=>10000 , min\_salary=>500);

* **FUNCTIONS**
* **Are pretty similar with the procedures**
* **Functions are get In and OUT parameters**
* **Functions must Return value**
* **Functions can be used with SELECT statement**
* **U can assign function to a variable**
* **Creating and Using Functions(SYNTAX)**

**CREATE [or REPLACE] function function\_name[(parameter\_name [IN|OUT] type DEFAULT value | expression [,…..])] RETURN return\_type {IS | AS}**

* **DIFFERENCES AND SIMILARITIES BETWEEN FUNCTIONS AND PROCEDURES**
  + **Procedures are executed with Begin/End block or executed with execute command**
  + **Functions are used with some SQL query or assigned to some variable**
  + **We can pass IN or OUT parameters to both**
  + **Procedures does not return value or functions must return value**
* **RESTRICTIONS IN USING FUNCTIONS**
  + **Must be compiled and stored in our database**
  + **Your functions should not an OUT parameter**
  + **Must return a valid types of the sql datatype**
  + **Cannot be used in table creation methods**
  + **Cannot call a function that contains a DML statement**
  + **Cannot include COMMIT, ROLLBACK or DDL commands**
* **FUNCTIONS EXAMPLES**
* CREATE OR REPLACE FUNCTION get\_avg\_sal (p\_dept\_id departments.department\_id%type) RETURN number AS

v\_avg\_sal number;

BEGIN

select avg(salary) into v\_avg\_sal from employees where department\_id = p\_dept\_id;

RETURN v\_avg\_sal;

END get\_avg\_sal;

* **using a function in begin-end block**

declare

v\_avg\_salary number;

begin

v\_avg\_salary := get\_avg\_sal(50);

dbms\_output.put\_line(v\_avg\_salary);

end;

* **using functions in a select clause**

select employee\_id,first\_name,salary,department\_id,get\_avg\_sal(department\_id) avg\_sal from employees;

* **using functions in group by, order by, where clauses**

select get\_avg\_sal(department\_id) from employees

where salary > get\_avg\_sal(department\_id)

group by get\_avg\_sal(department\_id)

order by get\_avg\_sal(department\_id);

* **dropping a function**

drop function get\_avg\_sal;

* **LOCAL SUBPROGRAMS**
* We can create a subprogram inside of an subprogram or an anonymous block
* Benefits
  + Reduce the code repetition
  + Improve the code readability
  + Need no more privilege
* But they are accessible only in the blocks they are defined
* We cannot create a subprogram in begin/end block
* We have to write the subprogram as the last in the declaration area
* **creating and using subprograms in anonymous blocks - false usage**

create table emps\_high\_paid as select \* from employees where 1=2;

/

declare

procedure insert\_high\_paid\_emp(emp\_id employees.employee\_id%type) is

emp employees%rowtype;

begin

emp := get\_emp(emp\_id);

insert into emps\_high\_paid values emp;

end;

function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

emp employees%rowtype;

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

end;

begin

for r\_emp in (select \* from employees) loop

if r\_emp.salary > 15000 then

insert\_high\_paid\_emp(r\_emp.employee\_id);

end if;

end loop;

end;

* **creating and using subprograms in anonymous blocks - true usage**

declare

function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

emp employees%rowtype;

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

end;

procedure insert\_high\_paid\_emp(emp\_id employees.employee\_id%type) is

emp employees%rowtype;

begin

emp := get\_emp(emp\_id);

insert into emps\_high\_paid values emp;

end;

begin

for r\_emp in (select \* from employees) loop

if r\_emp.salary > 15000 then

insert\_high\_paid\_emp(r\_emp.employee\_id);

end if;

end loop;

end;

* **The scope of emp record**

declare

procedure insert\_high\_paid\_emp(emp\_id employees.employee\_id%type) is

emp employees%rowtype;

e\_id number;

function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

end;

begin

emp := get\_emp(emp\_id);

insert into emps\_high\_paid values emp;

end;

begin

for r\_emp in (select \* from employees) loop

if r\_emp.salary > 15000 then

insert\_high\_paid\_emp(r\_emp.employee\_id);

end if;

end loop;

end;

* **OVERLOADING THE SUBPROGRAMS**
* **Overloading the subprograms with same name but with different parameters**
* **Overloading means creating more than one subprogram with same name**
* **Overloading is pretty useful when creating packages**
* **Benefits**
  + **Enables creating two or three subprograms with same name**
  + **So we can build more flexible subprograms**
  + **We can overload local and package subprograms but not standalone subprograms**
  + **Parameters must be different in data types or orders or numbers**
  + **Differentiating only the return type will not help**
* declare

procedure insert\_high\_paid\_emp(p\_emp employees%rowtype) is

emp employees%rowtype;

e\_id number;

function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

end;

function get\_emp(emp\_email employees.email%type) return employees%rowtype is

begin

select \* into emp from employees where email = emp\_email;

return emp;

end;

begin

emp := get\_emp(p\_emp.employee\_id);

insert into emps\_high\_paid values emp;

end;

begin

for r\_emp in (select \* from employees) loop

if r\_emp.salary > 15000 then

insert\_high\_paid\_emp(r\_emp);

end if;

end loop;

end;

* **overloading with multiple usages**

declare

procedure insert\_high\_paid\_emp(p\_emp employees%rowtype) is

emp employees%rowtype;

e\_id number;

function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

end;

function get\_emp(emp\_email employees.email%type) return employees%rowtype is

begin

select \* into emp from employees where email = emp\_email;

return emp;

end;

function get\_emp(f\_name varchar2, l\_name varchar2) return employees%rowtype is

begin

select \* into emp from employees where first\_name = f\_name and last\_name = l\_name;

return emp;

end;

begin

emp := get\_emp(p\_emp.employee\_id);

insert into emps\_high\_paid values emp;

emp := get\_emp(p\_emp.email);

insert into emps\_high\_paid values emp;

emp := get\_emp(p\_emp.first\_name,p\_emp.last\_name);

insert into emps\_high\_paid values emp;

end;

begin

for r\_emp in (select \* from employees) loop

if r\_emp.salary > 15000 then

insert\_high\_paid\_emp(r\_emp);

end if;

end loop;

end;

* **HANDLING EXCEPTIONS IN SUBPROGRAMS**
* **An unhandled exception in function**

create or replace function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

emp employees%rowtype;

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

end;

* **calling that function in an anonymous block**

declare

v\_emp employees%rowtype;

begin

dbms\_output.put\_line('Fetching the employee data!..');

v\_emp := get\_emp(10);

dbms\_output.put\_line('Some information of the employee are : ');

dbms\_output.put\_line('The name of the employee is : '|| v\_emp.first\_name);

dbms\_output.put\_line('The email of the employee is : '|| v\_emp.email);

dbms\_output.put\_line('The salary of the employee is : '|| v\_emp.salary);

end;

* **handling the exception wihout the return clause - not working**

create or replace function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

emp employees%rowtype;

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the id '|| emp\_num);

end;

* **handling and raising the exception**

create or replace function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

emp employees%rowtype;

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the id '|| emp\_num);

raise no\_data\_found;

end;

* **handling all possible exception cases**

create or replace function get\_emp(emp\_num employees.employee\_id%type) return employees%rowtype is

emp employees%rowtype;

begin

select \* into emp from employees where employee\_id = emp\_num;

return emp;

exception

when no\_data\_found then

dbms\_output.put\_line('There is no employee with the id '|| emp\_num);

raise no\_data\_found;

when others then

dbms\_output.put\_line('Something unexpected happened!.');

return null;

end;

* **REGULAR OR PIPELINED TABLE FUNCTIONS**
* Table functions return a table of varray or nested tables
* Regular table functions returns after completing whole data
* Pipelined functions returns each row one by one
* CREATE TYPE t\_day AS OBJECT (

v\_date DATE,

v\_day\_number INT

);

* **creating a nested table type**

CREATE TYPE t\_days\_tab IS TABLE OF t\_day;

* **creating a regular table function**

CREATE OR REPLACE FUNCTION f\_get\_days(p\_start\_date DATE , p\_day\_number INT)

RETURN t\_days\_tab IS

v\_days t\_days\_tab := t\_days\_tab();

BEGIN

FOR i IN 1 .. p\_day\_number LOOP

v\_days.EXTEND();

v\_days(i) := t\_day(p\_start\_date + i, to\_number(to\_char(p\_start\_date + i, 'DDD')));

END LOOP;

RETURN v\_days;

END;

* **querying from the regular table function**

select \* from table(f\_get\_days(sysdate,1000000));

* **querying from the regular table function without the table operator**

select \* from f\_get\_days(sysdate,1000000);

* **creating a pipelined table function**

create or replace function f\_get\_days\_piped (p\_start\_date date , p\_day\_number int)

return t\_days\_tab PIPELINED is

begin

for i in 1 .. p\_day\_number loop

PIPE ROW (t\_day(p\_start\_date + i,

to\_number(to\_char(p\_start\_date + i,'DDD'))));

end loop;

RETURN;

end;

* **querying from the pipelined table function**

select \* from f\_get\_days\_piped(sysdate,1000000)

* **PACKAGES**
* **Most of the times our objects work together**
* **There will be an object crowd in real work in time**
* **Packages can groups subprograms, types, variables etc in one container**
* **Package consists of two parts: Package specification and Package Body**
* **In package specification, we can only declare objects and all of them are public**
* **Package body has the implementations of all the subprograms declared in package specification**
* **Only Package body can use the objects declared in package specification**
* **Body and Spec should have the same name**
* **There is no Begin keyword in package**
* **Create or replace package package\_name as …. End package;**
* **CREATING & USING PACKAGES**
* Creating first package specification

CREATE OR REPLACE

PACKAGE EMP AS

v\_salary\_increase\_rate number := 0.057;

cursor cur\_emps is select \* from employees;

procedure increase\_salaries;

function get\_avg\_sal(p\_dept\_id int) return number;

END EMP;

* **Creating the package body**

CREATE OR REPLACE

PACKAGE BODY EMP AS

procedure increase\_salaries AS

BEGIN

for r1 in cur\_emps loop

update employees\_copy set salary = salary + salary \* v\_salary\_increase\_rate;

end loop;

END increase\_salaries;

function get\_avg\_sal(p\_dept\_id int) return number AS

v\_avg\_sal number := 0;

BEGIN

select avg(salary) into v\_avg\_sal from employees\_copy where

department\_id = p\_dept\_id;

RETURN v\_avg\_sal;

END get\_avg\_sal;

END EMP;

* **using the subprograms in packages**

exec EMP\_PKG.increase\_salaries;

* **using the variables in packages**

begin

dbms\_output.put\_line(emp\_pkg.get\_avg\_sal(50));

dbms\_output.put\_line(emp\_pkg.v\_salary\_increase\_rate);

end;

* **VISIBILITY OF VARIABLES IN PACKAGES**
* **create or replace PACKAGE BODY EMP\_PKG AS**

v\_sal\_inc int := 500;

v\_sal\_inc2 int := 500;

procedure print\_test is

begin

dbms\_output.put\_line('Test : '|| v\_sal\_inc);

end;

procedure increase\_salaries AS

BEGIN

for r1 in cur\_emps loop

update employees\_copy set salary = salary + salary \* v\_salary\_increase\_rate

where employee\_id = r1.employee\_id;

end loop;

END increase\_salaries;

function get\_avg\_sal(p\_dept\_id int) return number AS

v\_avg\_sal number := 0;

BEGIN

print\_test;

select avg(salary) into v\_avg\_sal from employees\_copy where

department\_id = p\_dept\_id;

RETURN v\_avg\_sal;

END get\_avg\_sal;

END EMP\_PKG;

* create or replace PACKAGE BODY EMP\_PKG AS

v\_sal\_inc int := 500;

v\_sal\_inc2 int := 500;

function get\_sal(e\_id employees.employee\_id%type) return number;

procedure print\_test is

begin

dbms\_output.put\_line('Test : '|| v\_sal\_inc);

dbms\_output.put\_line('Test salary : '|| get\_sal(102));

end;

procedure increase\_salaries AS

BEGIN

for r1 in cur\_emps loop

update employees\_copy set salary = salary + salary \* v\_salary\_increase\_rate

where employee\_id = r1.employee\_id;

end loop;

END increase\_salaries;

function get\_avg\_sal(p\_dept\_id int) return number AS

v\_avg\_sal number := 0;

BEGIN

print\_test;

select avg(salary) into v\_avg\_sal from employees\_copy where

department\_id = p\_dept\_id;

RETURN v\_avg\_sal;

END get\_avg\_sal;

function get\_sal(e\_id employees.employee\_id%type) return number is

v\_sal number := 0;

begin

select salary into v\_sal from employees where employee\_id = e\_id;

end;

end;

* **PERSISTENT STATE OF PACKAGES**
  + execute dbms\_output.put\_line(constants\_pkg.v\_salary\_increase);

grant execute on constants\_pkg to my\_user;

revoke execute on constants\_pkg from my\_user;

* begin

constants\_pkg.v\_company\_name := 'ACME';

dbms\_output.put\_line(constants\_pkg.v\_company\_name);

dbms\_lock.sleep(20);

end;

exec dbms\_output.put\_line(constants\_pkg.v\_company\_name);

* create or replace package constants\_pkg is

PRAGMA SERIALLY\_REUSABLE;

v\_salary\_increase constant number:= 0.04;

cursor cur\_emps is select \* from employees;

t\_emps\_type employees%rowtype;

v\_company\_name varchar2(20) := 'ORACLE';

end;

* begin

constants\_pkg.v\_company\_name := 'ACME';

dbms\_output.put\_line(constants\_pkg.v\_company\_name);

dbms\_lock.sleep(20);

end;

* declare

v\_emp employees%rowtype;

begin

open constants\_pkg.cur\_emps;

fetch constants\_pkg.cur\_emps into v\_emp;

dbms\_output.put\_line(v\_emp.first\_name);

close constants\_pkg.cur\_emps;

end;

* declare

v\_emp employees%rowtype;

begin

fetch constants\_pkg.cur\_emps into v\_emp;

dbms\_output.put\_line(v\_emp.first\_name);

end;

* **USING COLLECTIONS IN PACKAGE**
* create or replace PACKAGE EMP\_PKG AS

type emp\_table\_type is table of employees%rowtype index by pls\_integer;

v\_salary\_increase\_rate number := 1000;

v\_min\_employee\_salary number := 5000;

cursor cur\_emps is select \* from employees;

procedure increase\_salaries;

function get\_avg\_sal(p\_dept\_id int) return number;

v\_test int := 4;

function get\_employees return emp\_table\_type;

function get\_employees\_tobe\_incremented return emp\_table\_type;

procedure increase\_low\_salaries;

function arrange\_for\_min\_salary(v\_emp employees%rowtype) return employees%rowtype;

END EMP\_PKG;

* **package body**

create or replace PACKAGE BODY EMP\_PKG AS

v\_sal\_inc int := 500;

v\_sal\_inc2 int := 500;

function get\_sal(e\_id employees.employee\_id%type) return number;

procedure print\_test is

begin

dbms\_output.put\_line('Test : '|| v\_sal\_inc);

dbms\_output.put\_line('Tests salary :'|| get\_sal(102));

end;

procedure increase\_salaries AS

BEGIN

for r1 in cur\_emps loop

update employees\_copy set salary = salary + salary \* v\_salary\_increase\_rate

where employee\_id = r1.employee\_id;

end loop;

END increase\_salaries;

function get\_avg\_sal(p\_dept\_id int) return number AS

v\_avg\_sal number := 0;

BEGIN

print\_test;

select avg(salary) into v\_avg\_sal from employees\_copy where

department\_id = p\_dept\_id;

RETURN v\_avg\_sal;

END get\_avg\_sal;

function get\_sal(e\_id employees.employee\_id%type) return number is

v\_sal number := 0;

begin

select salary into v\_sal from employees where employee\_id = e\_id;

return v\_sal;

end;

/\*

This function returns all the employees in employees table

\*/

function get\_employees return emp\_table\_type is

v\_emps emp\_table\_type;

begin

for cur\_emps in (select \* from employees\_copy) loop

v\_emps(cur\_emps.employee\_id) := cur\_emps;

end loop;

return v\_emps;

end;

/\*

This function returns the employees which are under the minimum salary

of the company standards and to be incremented by the new minimum salary

\*/

function get\_employees\_tobe\_incremented return emp\_table\_type is

v\_emps emp\_table\_type;

i employees.employee\_id%type;

begin

v\_emps := get\_employees;

i := v\_emps.first;

while i is not null loop

if v\_emps(i).salary > v\_min\_employee\_salary then

v\_emps.delete(i);

end if;

i := v\_emps.next(i);

end loop;

return v\_emps;

end;

/\*

This procedure increases the salary of the employees who has a less salary

then the company standard

\*/

procedure increase\_low\_salaries as

v\_emps emp\_table\_type;

v\_emp employees%rowtype;

i employees.employee\_id%type;

begin

v\_emps := get\_employees\_tobe\_incremented;

i := v\_emps.first;

while i is not null loop

v\_emp := arrange\_for\_min\_salary(v\_emps(i));

update employees\_copy set row = v\_emp

where employee\_id = i;

i := v\_emps.next(i);

end loop;

end increase\_low\_salaries;

/\*

This function returns the employee by arranging the salary based on the

company standard.

\*/

function arrange\_for\_min\_salary(v\_emp in out employees%rowtype) return employees%rowtype is

begin

v\_emp.salary := v\_emp.salary + v\_salary\_increase\_rate;

if v\_emp.salary < v\_min\_employee\_salary then

v\_emp.salary := v\_min\_employee\_salary;

end if;

return v\_emp;

end;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BEGIN

v\_salary\_increase\_rate := 500;

insert into logs values ('EMP\_PKG','Package initialized!',sysdate);

END EMP\_PKG;

* **TRIGGERS**
* **Triggers are PL/SQL blocks that executed before or after or instead of a specific event**
* **Triggers are automatically triggered by database**
* **Triggers are defined on tables, views, schemas, databases**
* **Triggers are fired when one these occurs:**
* **When a DML(insert, update, delete) occurs**
* **When a DDL(create, alter, drop) occurs**
* **When some database operation occurs(server-on, start..)**
* **These are called database triggers**
* **Why do we use Triggers:**
  + **Security**
  + **Auditing**
  + **Data Integrity**
  + **Table Logging**
  + **Event Logging**
  + **Derived Data**
* **There are 3types of Triggers:**
  + **Compound Triggers**
  + **DML triggers**
  + **Non-DML triggers**
* **DML TRIGGERS**
* **Are PL/SQL blocks running when specific event occurs**
* **We use it for duplications, log maintenance**

**CREATE OR REPLACE TRIGGER TRIGGER\_NAME**

**TIMING = BEFORE | AFTER | INSTEAD OF**

**EVENT= INSERT | UPDATE | DELETE | UPDATE of column list**

**On object\_name**

**[REFERENCING OLD AS OLD OR NEW AS NEW]**

**[FOR EACH ROW]**

**[WHEN (condition)]**

**[ declare variables, types..]**

**BEGIN**

**Trigger body**

**[EXCEPTION]**

**END;**

* **SPECIFYING THE TIMING OF TRIGGERS**
* **The create code of the first trigger**

create or replace trigger first\_trigger

before insert or update on employees\_copy

begin

dbms\_output.put\_line('An insert or update occurred in employees\_copy table!.');

end;

* **sql commands to or not to run the trigger**

update employees\_copy set salary = salary + 100;

delete from employees\_copy;

* **STATEMENT & ROW LEVEL TRIGGERS**
* **before statement level trigger example**

create or replace trigger before\_statement\_emp\_cpy

before insert or update on employees\_copy

begin

dbms\_output.put\_line('Before Statement Trigger is Fired!.');

end;

* **after statement level trigger example**

create or replace trigger after\_statement\_emp\_cpy

after insert or update on employees\_copy

begin

dbms\_output.put\_line('After Statement Trigger is Fired!.');

end;

* **before row level trigger example**

create or replace trigger before\_row\_emp\_cpy

before insert or update on employees\_copy

for each row

begin

dbms\_output.put\_line('Before Row Trigger is Fired!.');

end;

* **after row level trigger example**

create or replace trigger after\_row\_emp\_cpy

after insert or update on employees\_copy

for each row

begin

dbms\_output.put\_line('After Row Trigger is Fired!.');

end;

* **sql queries used in this lecture**

update employees\_copy set salary = salary + 100 where employee\_id = 100;

update employees\_copy set salary = salary + 100 where employee\_id = 99;

update employees\_copy set salary = salary + 100

where department\_id = 30;

* **:NEW & :OLD QUALIFIERS IN TRIGGERS**

create or replace trigger before\_row\_emp\_cpy

before insert or update or delete on employees\_copy

referencing old as O new as N

for each row

begin

dbms\_output.put\_line('Before Row Trigger is Fired!.');

dbms\_output.put\_line('The Salary of Employee '||:o.employee\_id

||' -> Before:'|| :o.salary||' After:'||:n.salary);

* **USING CONDITIONAL PREDICATES**

create or replace trigger before\_row\_emp\_cpy

before insert or update or delete on employees\_copy

referencing old as O new as N

for each row

begin

dbms\_output.put\_line('Before Row Trigger is Fired!.');

dbms\_output.put\_line('The Salary of Employee '||:o.employee\_id

||' -> Before:'|| :o.salary||' After:'||:n.salary);

if inserting then

dbms\_output.put\_line('An INSERT occurred on employees\_copy table');

elsif deleting then

dbms\_output.put\_line('A DELETE occurred on employees\_copy table');

elsif updating ('salary') then

dbms\_output.put\_line('A DELETE occurred on the salary column');

elsif updating then

dbms\_output.put\_line('An UPDATE occurred on employees\_copy table');

end if;

end;

* **USING RAISE\_APPLICATION\_ERROR PROCEDURE WITH TRIGGERS**

create or replace trigger before\_row\_emp\_cpy

before insert or update or delete on employees\_copy

referencing old as O new as N

for each row

begin

dbms\_output.put\_line('Before Row Trigger is Fired!.');

dbms\_output.put\_line('The Salary of Employee '||:o.employee\_id

||' -> Before:'|| :o.salary||' After:'||:n.salary);

if inserting then

if :n.hire\_date > sysdate then

raise\_application\_error(-20000,'You cannot enter a future hire..');

end if;

elsif deleting then

raise\_application\_error(-20001,'You cannot delete from the employees\_copy table..');

elsif updating ('salary') then

if :n.salary > 50000 then

raise\_application\_error(-20002,'A salary cannot be higher than 50000..');

end if;

elsif updating then

dbms\_output.put\_line('An UPDATE occurred on employees\_copy table');

end if;

end;

* **USING UPDATE OF EVENT IN TRIGGERS**

create or replace trigger prevent\_updates\_of\_constant\_columns

before update of hire\_date,salary on employees\_copy

for each row

begin

raise\_application\_error(-20005,'You cannot modify the hire\_date and salary columns');

end;

* **USING WHEN CLAUSE ON TRIGGERS**

create or replace trigger prevent\_high\_salary

before insert or update of salary on employees\_copy

for each row

when (new.salary > 50000)

begin

raise\_application\_error(-20006,'A salary cannot be higher than 50000!.');

end;

* **USING INSTEAD OF TRIGGERS**
* **creating a complex view**

CREATE OR REPLACE VIEW VW\_EMP\_DETAILS AS

SELECT UPPER(DEPARTMENT\_NAME) DNAME, MIN(SALARY) MIN\_SAL, MAX(SALARY) MAX\_SAL

FROM EMPLOYEES\_COPY JOIN DEPARTMENTS\_COPY

USING (DEPARTMENT\_ID)

GROUP BY DEPARTMENT\_NAME;

* **updating the complex view -----------------**

UPDATE VW\_EMP\_DETAILS SET DNAME = 'EXEC DEPT' WHERE

UPPER(DNAME) = 'EXECUTIVE';

* **Instead of trigger -----------------**

CREATE OR REPLACE TRIGGER EMP\_DETAILS\_VW\_DML

INSTEAD OF INSERT OR UPDATE OR DELETE ON VW\_EMP\_DETAILS

FOR EACH ROW

DECLARE

V\_DEPT\_ID PLS\_INTEGER;

BEGIN

IF INSERTING THEN

SELECT MAX(DEPARTMENT\_ID) + 10 INTO V\_DEPT\_ID FROM DEPARTMENTS\_COPY;

INSERT INTO DEPARTMENTS\_COPY VALUES (V\_DEPT\_ID, :NEW.DNAME,NULL,NULL);

ELSIF DELETING THEN

DELETE FROM DEPARTMENTS\_COPY WHERE UPPER(DEPARTMENT\_NAME) = UPPER(:OLD.DNAME);

ELSIF UPDATING('DNAME') THEN

UPDATE DEPARTMENTS\_COPY SET DEPARTMENT\_NAME = :NEW.DNAME

WHERE UPPER(DEPARTMENT\_NAME) = UPPER(:OLD.DNAME);

ELSE

RAISE\_APPLICATION\_ERROR(-20007,'You cannot update any data other than department name!.');

END IF;

END;

* **CREATING DISABLED TRIGGERS**

create or replace trigger prevent\_high\_salary

before insert or update of salary on employees\_copy

for each row

disable

when (new.salary > 50000)

begin

raise\_application\_error(-20006,'A salary cannot be higher than 50000!.');

end;

* **REAL-WORLD EXAMPLES ON DML TRIGGERS**

create sequence seq\_dep\_cpy

start with 280

increment by 10;

* + **primary key example**

create or replace trigger trg\_before\_insert\_dept\_cpy

before insert on departments\_copy

for each row

begin

--select seq\_dep\_cpy.nextval into :new.department\_id from dual;

:new.department\_id := seq\_dep\_cpy.nextval;

end;

-----------------

insert into departments\_copy

(department\_name,manager\_id,location\_id)

values

('Security',200,1700);

-----------------

desc departments\_copy;

* + **creating the audit log table**

create table log\_departments\_copy

(log\_user varchar2(30), log\_date date, dml\_type varchar2(10),

old\_department\_id number(4), new\_department\_id number(4),

old\_department\_name varchar2(30), new\_department\_name varchar2(30),

old\_manager\_id number(6), new\_manager\_id number(6),

old\_location\_id number(4), new\_location\_id number(4));

* + **audit log trigger**

create or replace trigger trg\_department\_copy\_log

after insert or update or delete on departments\_copy

for each row

declare v\_dml\_type varchar2(10);

begin

if inserting then

v\_dml\_type := 'INSERT';

elsif updating then

v\_dml\_type := 'UPDATE';

elsif deleting then

v\_dml\_type := 'DELETE';

end if;

insert into log\_departments\_copy values

(user, sysdate, v\_dml\_type,

:old.department\_id, :new.department\_id,

:old.department\_name, :new.department\_name,

:old.manager\_id, :new.manager\_id,

:old.location\_id, :new.location\_id);

end;

* + **other sql codes used in this lecture**

insert into departments\_copy (department\_name, manager\_id,location\_id)

values ('Cyber Security', 100, 1700);

select \* from LOG\_DEPARTMENTS\_COPY;

update departments\_copy set manager\_id = 200 where DEPARTMENT\_NAME = 'Cyber Security';

delete from departments\_copy where DEPARTMENT\_NAME = 'Cyber Security';

* **COMPOUND TRIGGERS**
  + **Is a single trigger that allow us to specify actions for each DML trigger types**
  + **So they can share the variables, types amongst etc.**
  + **WHY WE USE:**
    - **Taking actions for some various timing points by sharing common data**
    - **Making inserts to some other tables much faster with bulk inserts**
    - **Avoiding mutating table inserts**
* **The first simple compound trigger**

create or replace trigger trg\_comp\_emps

for insert or update or delete on employees\_copy

compound trigger

v\_dml\_type varchar2(10);

before statement is

begin

if inserting then

v\_dml\_type := 'INSERT';

elsif updating then

v\_dml\_type := 'UPDATE';

elsif deleting then

v\_dml\_type := 'DELETE';

end if;

dbms\_output.put\_line('Before statement section is executed with the '||v\_dml\_type ||' event!.');

end before statement;

before each row is

t number;

begin

dbms\_output.put\_line('Before row section is executed with the '||v\_dml\_type ||' event!.');

end before each row;

after each row is

begin

dbms\_output.put\_line('After row section is executed with the '||v\_dml\_type ||' event!.');

end after each row;

after statement is

begin

dbms\_output.put\_line('After statement section is executed with the '||v\_dml\_type ||' event!.');

end after statement;

end;

* CREATE OR REPLACE TRIGGER TRG\_COMP\_EMPS

FOR INSERT OR UPDATE OR DELETE ON EMPLOYEES\_COPY

COMPOUND TRIGGER

TYPE T\_AVG\_DEPT\_SALARIES IS TABLE OF EMPLOYEES\_COPY.SALARY%TYPE INDEX BY PLS\_INTEGER;

AVG\_DEPT\_SALARIES T\_AVG\_DEPT\_SALARIES;

BEFORE STATEMENT IS

BEGIN

FOR AVG\_SAL IN (SELECT AVG(SALARY) SALARY , NVL(DEPARTMENT\_ID,999) DEPARTMENT\_ID

FROM EMPLOYEES\_COPY GROUP BY DEPARTMENT\_ID) LOOP

AVG\_DEPT\_SALARIES(AVG\_SAL.DEPARTMENT\_ID) := AVG\_SAL.SALARY;

END LOOP;

END BEFORE STATEMENT;

AFTER EACH ROW IS

V\_INTERVAL NUMBER := 15;

BEGIN

IF :NEW.SALARY > AVG\_DEPT\_SALARIES(:NEW.DEPARTMENT\_ID) + AVG\_DEPT\_SALARIES(:NEW.DEPARTMENT\_ID)\*V\_INTERVAL/100 THEN

RAISE\_APPLICATION\_ERROR(-20005,'A raise cannot be '|| V\_INTERVAL|| ' percent higher than

its department''s average!');

END IF;

END AFTER EACH ROW;

AFTER STATEMENT IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('All the changes are done successfully!');

END AFTER STATEMENT;

END;

* **DYNAMIC SQL & PL/SQL IN PL/SQL**

BEGIN

EXECUTE IMMEDIATE 'GRANT SELECT ON EMPLOYEES TO SYS';

END;

/

BEGIN

EXECUTE IMMEDIATE 'GRANT SELECT ON EMPLOYEES TO SYS;';

END;

/

CREATE OR REPLACE PROCEDURE prc\_create\_table\_dynamic

(p\_table\_name IN VARCHAR2, p\_col\_specs IN VARCHAR2) IS

BEGIN

EXECUTE IMMEDIATE 'CREATE TABLE '||p\_table\_name||' ('||p\_col\_specs||')';

END;

/

EXEC prc\_create\_table\_dynamic('dynamic\_temp\_table', 'id NUMBER PRIMARY KEY, name VARCHAR2(100)');

/

SELECT \* FROM dynamic\_temp\_table;

/

CREATE OR REPLACE PROCEDURE prc\_generic (p\_dynamic\_sql IN VARCHAR2) IS

BEGIN

EXECUTE IMMEDIATE p\_dynamic\_sql;

END;

/

EXEC prc\_generic('drop table dynamic\_temp\_table');

/

EXEC prc\_generic('drop procedure PRC\_CREATE\_TABLE\_DYNAMIC');

/

DROP PROCEDURE prc\_generic;

* **EXECUTE IMMEDIATE STATEMENT WITH THE USING CLAUSE**

CREATE TABLE names (ID NUMBER PRIMARY KEY, NAME VARCHAR2(100));

/

CREATE OR REPLACE FUNCTION insert\_values (ID IN VARCHAR2, NAME IN VARCHAR2) RETURN PLS\_INTEGER IS

BEGIN

EXECUTE IMMEDIATE 'INSERT INTO names VALUES(:a, :b)' USING ID,NAME;

RETURN SQL%rowcount;

END;

/

SET SERVEROUTPUT ON;

DECLARE

v\_affected\_rows PLS\_INTEGER;

BEGIN

v\_affected\_rows := insert\_values(2,'John');

dbms\_output.put\_line(v\_affected\_rows|| ' row inserted!');

END;

/

SELECT \* FROM names;

/

ALTER TABLE names ADD (last\_name VARCHAR2(100));

/

CREATE OR REPLACE FUNCTION update\_names (ID IN VARCHAR2, last\_name IN VARCHAR2) RETURN PLS\_INTEGER IS

v\_dynamic\_sql VARCHAR2(200);

BEGIN

v\_dynamic\_sql := 'UPDATE names SET last\_name = :1 WHERE id = :2' ;

EXECUTE IMMEDIATE v\_dynamic\_sql USING last\_name, ID;

RETURN SQL%rowcount;

END;

/

DECLARE

v\_affected\_rows PLS\_INTEGER;

BEGIN

v\_affected\_rows := update\_names(2,'Brown');

dbms\_output.put\_line(v\_affected\_rows|| ' row updated!');

END;

/

CREATE OR REPLACE FUNCTION update\_names (ID IN VARCHAR2, last\_name IN OUT VARCHAR2) RETURN PLS\_INTEGER IS

v\_dynamic\_sql VARCHAR2(200);

BEGIN

v\_dynamic\_sql := 'UPDATE names SET last\_name = :1 WHERE id = :2' ;

EXECUTE IMMEDIATE v\_dynamic\_sql USING IN OUT last\_name, ID;

RETURN SQL%rowcount;

END;

/

CREATE OR REPLACE FUNCTION update\_names (ID IN VARCHAR2, last\_name IN VARCHAR2, first\_name OUT VARCHAR2) RETURN PLS\_INTEGER IS

v\_dynamic\_sql VARCHAR2(200);

BEGIN

v\_dynamic\_sql := 'UPDATE names SET last\_name = :1 WHERE id = :2 :3' ;

EXECUTE IMMEDIATE v\_dynamic\_sql USING last\_name, ID, OUT first\_name;

RETURN SQL%rowcount;

END;

/

DECLARE

v\_affected\_rows PLS\_INTEGER;

v\_first\_name VARCHAR2(100);

BEGIN

v\_affected\_rows := update\_names(2,'KING',v\_first\_name);

dbms\_output.put\_line(v\_affected\_rows|| ' row updated!');

dbms\_output.put\_line(v\_first\_name);

END;

/

CREATE OR REPLACE FUNCTION update\_names (ID IN VARCHAR2, last\_name IN VARCHAR2, first\_name OUT VARCHAR2) RETURN PLS\_INTEGER IS

v\_dynamic\_sql VARCHAR2(200);

BEGIN

v\_dynamic\_sql := 'UPDATE names SET last\_name = :1 WHERE id = :2 RETURNING name INTO :3' ;

EXECUTE IMMEDIATE v\_dynamic\_sql USING last\_name, ID RETURNING INTO first\_name;

RETURN SQL%rowcount;

END;

/

DROP TABLE names;

DROP FUNCTION insert\_values;

DROP FUNCTION update\_names;

* **EXECUTE IMMEDIATE STATEMENT WITH THE USING AND INTO CLAUSES**

CREATE OR REPLACE FUNCTION get\_count (table\_name IN VARCHAR2) RETURN PLS\_INTEGER IS

v\_count PLS\_INTEGER;

BEGIN

EXECUTE IMMEDIATE 'SELECT COUNT(\*) FROM ' || table\_name INTO v\_count;

RETURN v\_count;

END;

/

SET SERVEROUTPUT ON;

BEGIN

dbms\_output.put\_line('There are '||get\_count('employees')||' rows in the employees table!');

END;

/

DECLARE

v\_table\_name VARCHAR2(50);

BEGIN

FOR r\_table IN (SELECT table\_name FROM user\_tables) LOOP

dbms\_output.put\_line('There are '||get\_count(r\_table.table\_name)||' rows in the '||r\_table.table\_name||' table!');

END LOOP;

END;

/

DECLARE

v\_table\_name VARCHAR2(50);

BEGIN

FOR r\_table IN (SELECT table\_name FROM user\_tables) LOOP

IF get\_count(r\_table.table\_name) > 100 THEN

dbms\_output.put\_line('There are '||get\_count(r\_table.table\_name)||' rows in the '||r\_table.table\_name||' table!');

dbms\_output.put\_line('It should be considered for partitioning');

END IF;

END LOOP;

END;

/

CREATE TABLE stock\_managers AS SELECT \* FROM employees WHERE job\_id = 'ST\_MAN';

/

CREATE TABLE stock\_clerks AS SELECT \* FROM employees WHERE job\_id = 'ST\_CLERK';

/

CREATE OR REPLACE FUNCTION get\_avg\_sals (p\_table IN VARCHAR2, p\_dept\_id IN NUMBER) RETURN PLS\_INTEGER IS

v\_average PLS\_INTEGER;

BEGIN

EXECUTE IMMEDIATE 'SELECT AVG(salary) FROM :1 WHERE department\_id = :2' INTO v\_average USING p\_table, p\_dept\_id;

RETURN v\_average;

END;

/

SELECT get\_avg\_sals('stock\_clerks','50') FROM dual;

/

CREATE OR REPLACE FUNCTION get\_avg\_sals (p\_table IN VARCHAR2, p\_dept\_id IN NUMBER) RETURN PLS\_INTEGER IS

v\_average PLS\_INTEGER;

BEGIN

EXECUTE IMMEDIATE 'SELECT AVG(salary) FROM '||p\_table||' WHERE department\_id = :2' INTO v\_average USING p\_dept\_id;

RETURN v\_average;

END;

/

SELECT get\_avg\_sals('stock\_managers','50') FROM dual;

/

DROP FUNCTION get\_count;

DROP FUNCTION get\_avg\_sals;

DROP TABLE stock\_clerks;

DROP TABLE stock\_managers;

* **EXECUTE IMMEDIATE WITH BULK COLLECT**

DECLARE

TYPE t\_name IS TABLE OF VARCHAR2(20);

names t\_name;

BEGIN

EXECUTE IMMEDIATE 'SELECT distinct first\_name FROM employees'

BULK COLLECT INTO names;

FOR i IN 1..names.COUNT LOOP

dbms\_output.put\_line(names(i));

END LOOP;

END;

/

CREATE TABLE employees\_copy AS SELECT \* FROM employees;

/

DECLARE

TYPE t\_name IS TABLE OF VARCHAR2(20);

names t\_name;

BEGIN

EXECUTE IMMEDIATE 'UPDATE employees\_copy SET salary = salary + 1000 WHERE department\_id = 30 RETURNING first\_name INTO :a'

RETURNING BULK COLLECT INTO names;

FOR i IN 1..names.COUNT LOOP

dbms\_output.put\_line(names(i));

END LOOP;

END;

/

DROP TABLE employees\_copy;

* **DYNAMIC PL/SQL BLOCKS**

BEGIN

FOR r\_emp in (SELECT \* FROM employees) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

END LOOP;

END;

/

DECLARE

v\_dynamic\_text varchar2(1000);

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employees) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text;

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

v\_department\_id PLS\_INTEGER := 30;

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employees WHERE department\_id = v\_department\_id) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text;

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

--v\_department\_id pls\_integer := 30;

BEGIN

v\_dynamic\_text := q'[DECLARE

v\_department\_id pls\_integer := 30;

BEGIN

FOR r\_emp in (SELECT \* FROM employees WHERE department\_id = v\_department\_id) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text;

END;

/

CREATE OR REPLACE PACKAGE pkg\_temp AS

v\_department\_id\_pkg PLS\_INTEGER := 50;

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

--v\_department\_id pls\_integer := 30;

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employees WHERE department\_id = pkg\_temp.v\_department\_id\_pkg) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text;

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

v\_department\_id PLS\_INTEGER := 30;

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employees WHERE department\_id = :1) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text USING v\_department\_id;

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

v\_department\_id PLS\_INTEGER := 30;

v\_max\_salary PLS\_INTEGER := 0;

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employees WHERE department\_id = :1) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

if r\_emp.salary > :sal then

:sal := r\_emp.salary;

end if;

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text USING v\_department\_id, IN OUT v\_max\_salary;

dbms\_output.put\_line('The maximum salary of this department is : '||v\_max\_salary);

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

v\_department\_id PLS\_INTEGER := 30;

v\_max\_salary PLS\_INTEGER := 0;

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employeese WHERE department\_id = :1) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

if r\_emp.salary > :sal then

:sal := r\_emp.salary;

end if;

END LOOP;

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text USING v\_department\_id, IN OUT v\_max\_salary;

dbms\_output.put\_line('The maximum salary of this department is : '||v\_max\_salary);

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('The error is : '||sqlerrm);

END;

/

DECLARE

v\_dynamic\_text VARCHAR2(1000);

v\_department\_id PLS\_INTEGER := 30;

v\_max\_salary PLS\_INTEGER := 0;

BEGIN

v\_dynamic\_text := q'[BEGIN

FOR r\_emp in (SELECT \* FROM employeese WHERE department\_id = :1) LOOP

dbms\_output.put\_line(r\_emp.first\_name||' '||r\_emp.last\_name);

if r\_emp.salary > :sal then

:sal := r\_emp.salary;

end if;

END LOOP;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('The error is : '||SQLERRM);

END;]';

EXECUTE IMMEDIATE v\_dynamic\_text USING v\_department\_id, IN OUT v\_max\_salary;

dbms\_output.put\_line('The maximum salary of this department is : '||v\_max\_salary);

END;

/

DROP PACKAGE pkg\_temp;

* **OPEN - FOR, FETCH STATEMENTS**

DECLARE

TYPE emp\_cur\_type IS REF CURSOR;

emp\_cursor emp\_cur\_type;

emp\_record employees%rowtype;

BEGIN

OPEN emp\_cursor FOR 'SELECT \* FROM employees WHERE job\_id = ''IT\_PROG''';

FETCH emp\_cursor INTO emp\_record;

dbms\_output.put\_line(emp\_record.first\_name||emp\_record.last\_name);

CLOSE emp\_cursor;

END;

/

DECLARE

TYPE emp\_cur\_type IS REF CURSOR;

emp\_cursor emp\_cur\_type;

emp\_record employees%rowtype;

BEGIN

OPEN emp\_cursor FOR 'SELECT \* FROM employees WHERE job\_id = :job' USING 'IT\_PROG';

FETCH emp\_cursor INTO emp\_record;

dbms\_output.put\_line(emp\_record.first\_name||emp\_record.last\_name);

CLOSE emp\_cursor;

END;

/

DECLARE

TYPE emp\_cur\_type IS REF CURSOR;

emp\_cursor emp\_cur\_type;

emp\_record employees%rowtype;

BEGIN

OPEN emp\_cursor FOR 'SELECT \* FROM employees WHERE job\_id = :job' USING 'IT\_PROG';

LOOP

FETCH emp\_cursor INTO emp\_record;

EXIT WHEN emp\_cursor%notfound;

dbms\_output.put\_line(emp\_record.first\_name||emp\_record.last\_name);

END LOOP;

CLOSE emp\_cursor;

END;

/

DECLARE

TYPE emp\_cur\_type IS REF CURSOR;

emp\_cursor emp\_cur\_type;

emp\_record employees%rowtype;

v\_table\_name VARCHAR(20);

BEGIN

v\_table\_name := 'employees';

OPEN emp\_cursor FOR 'SELECT \* FROM '||v\_table\_name||' WHERE job\_id = :job' USING 'IT\_PROG';

LOOP

FETCH emp\_cursor INTO emp\_record;

EXIT WHEN emp\_cursor%notfound;

dbms\_output.put\_line(emp\_record.first\_name||emp\_record.last\_name);

END LOOP;

CLOSE emp\_cursor;

END;

* **Using the DBMS\_SQL Package**

CREATE TABLE employees\_copy AS SELECT \* FROM employees;

/

set serveroutput on;

DECLARE

v\_table\_name VARCHAR2(20) := 'employees\_copy';

v\_cursor\_id PLS\_INTEGER;

v\_affected\_rows PLS\_INTEGER;

BEGIN

v\_cursor\_id := dbms\_sql.open\_cursor;

dbms\_sql.parse(v\_cursor\_id, 'update '||v\_table\_name||' set salary=salary+500',dbms\_sql.NATIVE);

v\_affected\_rows := dbms\_sql.EXECUTE(v\_cursor\_id);

dbms\_output.put\_line(v\_affected\_rows|| ' rows are updated by dbms\_sql!');

dbms\_sql.close\_cursor(v\_cursor\_id);

END;

/

select \* from employees\_copy;

/

DECLARE

v\_table\_name varchar2(20) := 'employees\_copy';

v\_cursor\_id pls\_integer;

v\_affected\_rows pls\_integer;

BEGIN

v\_cursor\_id := DBMS\_SQL.OPEN\_CURSOR;

DBMS\_SQL.PARSE(v\_cursor\_id, 'update '||v\_table\_name||' set salary=salary+500 WHERE job\_id = :jid',DBMS\_SQL.NATIVE);

DBMS\_SQL.BIND\_VARIABLE(v\_cursor\_id, ':jid','IT\_PROG');

v\_affected\_rows := DBMS\_SQL.EXECUTE(v\_cursor\_id);

dbms\_output.put\_line(v\_affected\_rows|| ' rows are updated by dbms\_sql!');

DBMS\_SQL.CLOSE\_CURSOR(v\_cursor\_id);

END;

 /

DECLARE

v\_table\_name varchar2(20) := 'employees\_copy';

v\_cursor\_id pls\_integer;

v\_affected\_rows pls\_integer;

BEGIN

v\_cursor\_id := DBMS\_SQL.OPEN\_CURSOR;

DBMS\_SQL.PARSE(v\_cursor\_id, 'update '||v\_table\_name||' set salary=salary+:inc WHERE job\_id = :jid',DBMS\_SQL.NATIVE);

DBMS\_SQL.BIND\_VARIABLE(v\_cursor\_id, ':jid','IT\_PROG');

DBMS\_SQL.BIND\_VARIABLE(v\_cursor\_id, ':inc','5');

v\_affected\_rows := DBMS\_SQL.EXECUTE(v\_cursor\_id);

dbms\_output.put\_line(v\_affected\_rows|| ' rows are updated by dbms\_sql!');

DBMS\_SQL.CLOSE\_CURSOR(v\_cursor\_id);

END;

/

SELECT \* FROM user\_tab\_columns;

EXEC prc\_method4\_example('employees');

EXEC prc\_method4\_example('departments');

EXEC prc\_method4\_example('countries');

EXEC prc\_method4\_example('locations');

/

create or replace PROCEDURE prc\_method4\_example (p\_table\_name IN VARCHAR2) IS

TYPE t\_columns IS TABLE OF user\_tab\_columns%rowtype INDEX BY PLS\_INTEGER;

v\_columns t\_columns;

v\_columns\_with\_commas VARCHAR2(32767);

v\_number\_value NUMBER;

v\_string\_value VARCHAR2(32767);

v\_date\_value DATE;

v\_output\_string VARCHAR2(32767);

cur\_dynamic INTEGER;

BEGIN

SELECT \* BULK COLLECT INTO v\_columns FROM user\_tab\_columns WHERE table\_name = upper(p\_table\_name);

v\_columns\_with\_commas:=v\_columns(1).column\_name;

FOR i IN 2..v\_columns.COUNT LOOP

v\_columns\_with\_commas:=v\_columns\_with\_commas||','||v\_columns(i).column\_name;

END LOOP;

cur\_dynamic := dbms\_sql.open\_cursor;

dbms\_sql.parse(cur\_dynamic,'SELECT '||v\_columns\_with\_commas||' FROM '||p\_table\_name,dbms\_sql.NATIVE);

FOR idx IN 1..v\_columns.COUNT LOOP

IF v\_columns(idx).data\_type = 'NUMBER' THEN

dbms\_sql.define\_column(cur\_dynamic,idx,1);

ELSIF v\_columns(idx).data\_type IN ('VARCHAR2','VARCHAR','CHAR') THEN

dbms\_sql.define\_column(cur\_dynamic,idx,'dummy text',v\_columns(idx).char\_length);

ELSIF v\_columns(idx).data\_type = 'DATE' THEN

dbms\_sql.define\_column(cur\_dynamic,idx,sysdate);

END IF;

v\_output\_string:=v\_output\_string||' '||rpad(v\_columns(idx).column\_name,20);

END LOOP;

dbms\_output.put\_line(v\_output\_string);

v\_number\_value:=dbms\_sql.execute(cur\_dynamic);

WHILE dbms\_sql.fetch\_rows(cur\_dynamic) > 0 LOOP

v\_output\_string:=NULL;

FOR t IN 1..v\_columns.COUNT LOOP

IF v\_columns(T).data\_type = 'NUMBER' THEN

dbms\_sql.column\_value(cur\_dynamic,t,v\_number\_value);

v\_output\_string := v\_output\_string||' '||rpad(nvl(to\_char(v\_number\_value),' '),20);

ELSIF v\_columns(T).data\_type IN ('VARCHAR2','VARCHAR','CHAR') THEN

dbms\_sql.column\_value(cur\_dynamic,t,v\_string\_value);

v\_output\_string := v\_output\_string||' '||rpad(nvl(to\_char(v\_string\_value),' '),20);

ELSIF v\_columns(T).data\_type = 'DATE' THEN

dbms\_sql.column\_value(cur\_dynamic,t,v\_date\_value);

v\_output\_string := v\_output\_string||' '||rpad(nvl(to\_char(v\_date\_value),' '),20);

END IF;

END LOOP;

dbms\_output.put\_line(v\_output\_string);

END LOOP;

END;