Blue means optional

# Anonymous Block

Declare

{variable name} {data type} [:= {value}];

Begin

{body code}

End;

# Cursors

Using for loop:

declare

cursor c is select \* from t;

begin

for i in c loop

if(i.sid = 1) then

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

end if;

end loop;

end;

Declare

Cursor {cursor name} is {select statement};

Begin

For {variable name} in {cursor name} loop

{loop body}

[if {Boolean statement} then {execution code} [elsif…] end if]

End loop;

End;

Using normal loop:

declare

cursor c is select sid,name from t;

id t.sid%type;

name t.name%type;

begin

open c;

loop

fetch c into id,name;

exit when c%notfound;

if(id = 1) then

dbms\_output.put\_line(name);

end if;

end loop;

close c;

end;

Declare

Cursor {cursor name} is {select statement};

{variable name} {data type} ;

Begin

Open {cursor name};

Loop

Fetch {cursor name} into {variable name(s)};

Exit when {cursor name}%notfound;

End loop;

Close {cursor name};

End;

# Select Into

declare

r t%rowtype;

begin

select \* into r from t where sid = 1;

dbms\_output.put\_line(r.name);

end;

Declare

{variable name} {data type};

Begin

select {columns} into {variables} from {table name} {where clause};

dbms\_output.put\_line({variable name}.{column name});

End;

For-Loop

begin

for i in 1..10 loop

dbms\_output.put\_line(i);

end loop;

end;

Begin

For {variable name} in {start}..{end} loop

{execution body};

End loop;

End;

Exception handling: CURSOR\_ALREAY\_OPEN, INVALID\_CURSOR, NO\_DATA\_FOUND,

declare

negative\_num exception;

temp int := -5;

begin

if(temp < 0) then

raise negative\_num;

end if;

exception

when negative\_num then

dbms\_output.put\_line('negative num has been input');

end;

TOO\_MANY\_ROWS, ZERO\_DIVIDE, VALUE\_ERROR,OTHERS

Declare

{variable name} exception;

begin

if (Boolean statement) then raise {variable name}

end;

Triggers (‘instead of’ can only be used on views and not tables)

create or replace trigger print\_sal\_diff

after update on t

for each row

begin

if(:old.salary < :new.salary) then

dbms\_output.put\_line((:new.salary - :old.salary));

end if;

end;

create or replace trigger {trigger name} {before ,after, instead of}

{insert, update, delete} on {table/view name}

For each row

Declare

{declarations}

Begin

{body code}

End;

Records

declare

type r\_set is record(

temp\_id int,

temp\_name varchar(20)

);

r r\_set;

begin

select s\_id,first\_name into r from t where s\_id = 1;

dbms\_output.put\_line(r.temp\_id || r.temp\_name);

end;

Type {type name} is record(

{variable name} {datatype}

);

Functions (in keyword means read only and out keyword means output only)

create or replace function say\_hi

return varchar

as

begin

return 'lol';

end;

select say\_hi from dual;

Create or replace function {function name} [({parameters})]

return {data type}

as

{declarations}

Begin

{execution code with a return statement}

End;

Procedures

create or replace procedure say\_lol

as

begin

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

end;

exec say\_lol;

Create or replace procedure {procedure name} [({parameters})]

as

{declarations}

Begin

{execution code with a return statement}

End;

Arrays

declare

type arr is table of varchar(20) index by binary\_integer;

temp arr;

ind int;

begin

temp(1) := 'majd';

temp(2) := 'sbi';

ind := temp.first;

loop

exit when ind is null;

dbms\_output.put\_line(temp(ind));

ind := temp.next(ind);

end loop;

end;

Declare

Type {type name} is table of {values data type} index by {key data type}

{variable name} {type name};

Begin

{execution code};

End;

Packages

create or replace package test\_pack as

type test\_record is record(

fname varchar(20),

sal int

);

function getPerson(temp\_id in int) return test\_record;

end;

create or replace package body test\_pack as

function getPerson(temp\_id in int) return test\_record

as

temp test\_record;

begin

select first\_name,salary into temp from t where s\_id = temp\_id;

return temp;

end;

end;

begin

dbms\_output.put\_line(test\_pack.getPerson(1).sal);

end;

Create or replace package {package name} as

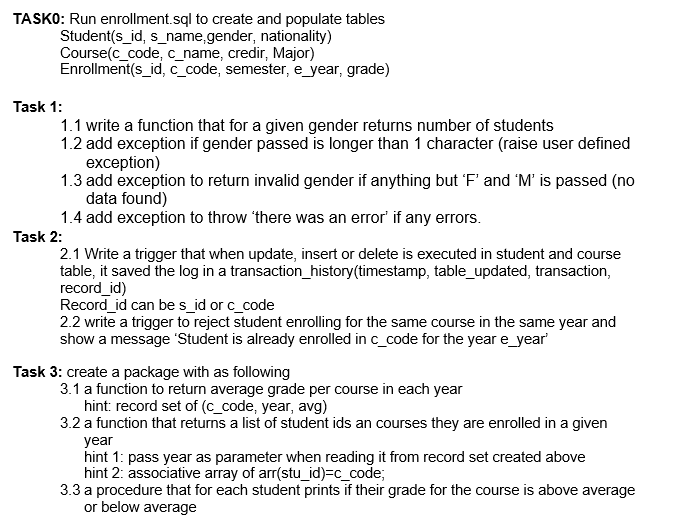
{what to include inside the package}

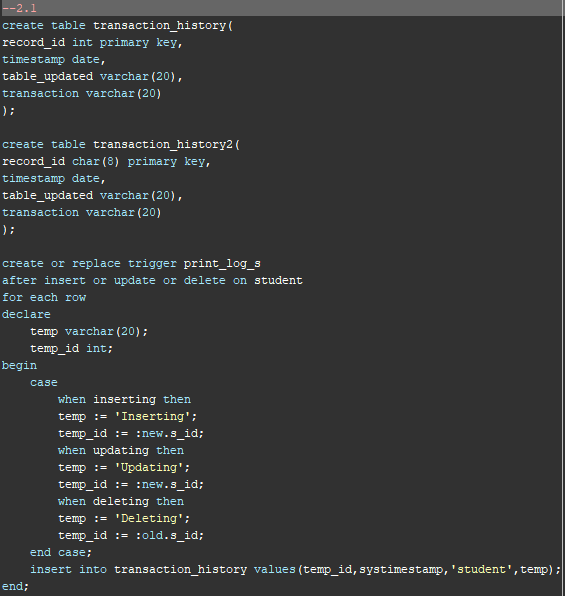
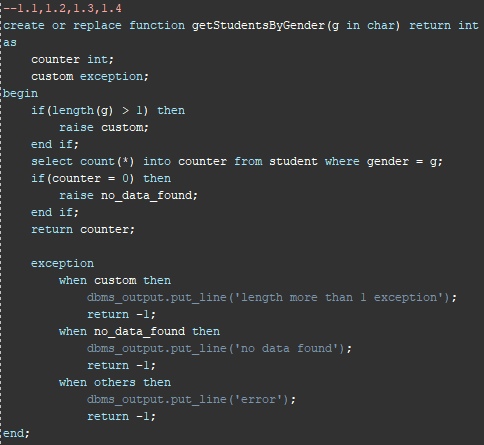
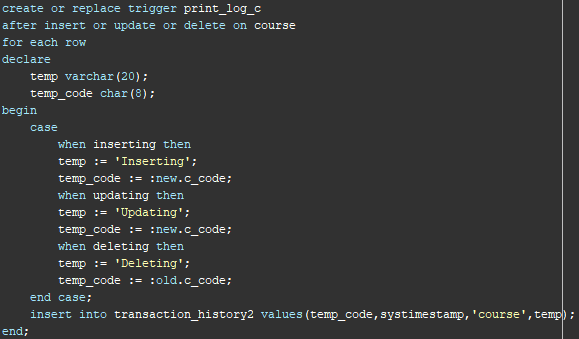
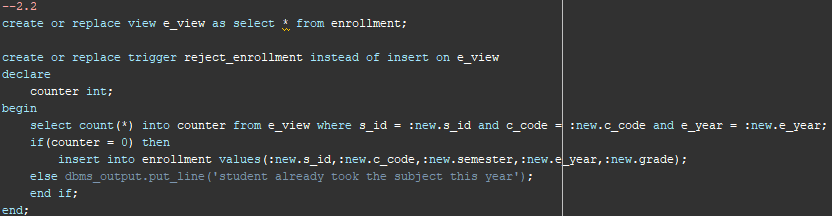
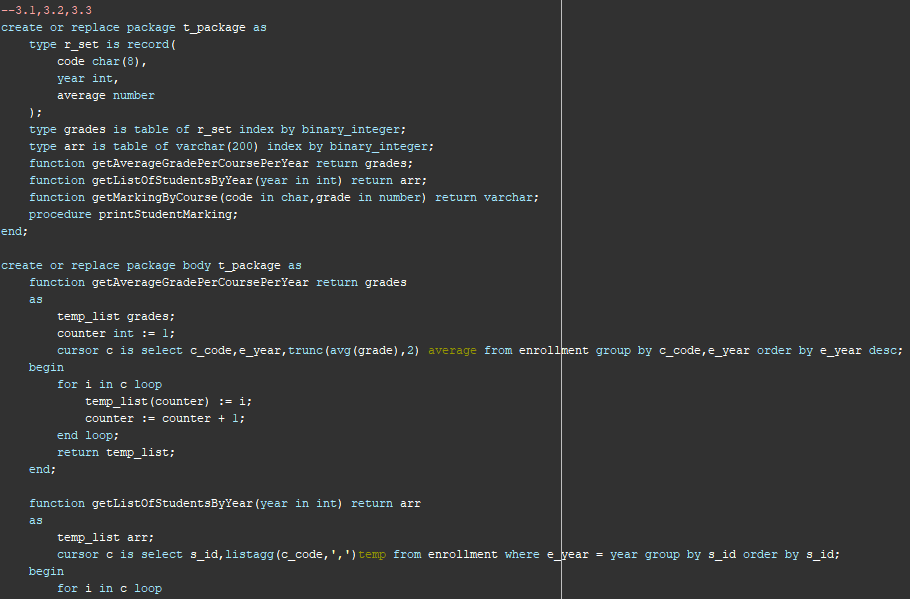
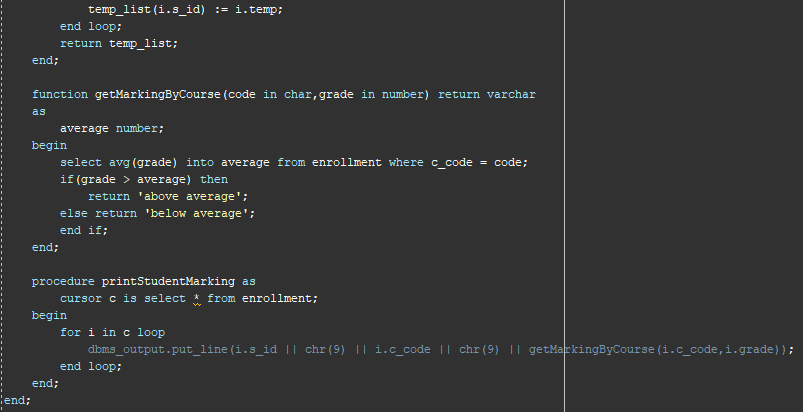
End;

Create or replace package body {package name} as

{body code for the functions and procedures}

End;



Functional Dependencies: is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. x→y

Armstrong axioms:

1) **Reflexivity:** If Y is a subset of X, then X→Y holds by reflexivity rule

2) **Augmentation:** If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.

3) **Transitivity**: If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.

4) **Union**: If X → Y and X → Z then X → YZ is also vaild by the union rule.

5) **Decomposition**: If X → YZ then X → Y and X → Z by the decomposition rule.

Attribute closure: is a subset that contains all the attributes that rely on it.

Set F(a,b,c,d,e) [a → b , b → c, c → de, cd → ab] a+ = {a,b,c,d}.

Decomposition: is the process of breaking up or dividing a single relation into two or more sub relations.

R (A, B, C, D) with functional dependency set (A→BC) (A →D) R is decomposed into R1(ABC) and

R2(AD).

Problems of Decompositions:

1) some queries become more expensive.

2) Decomposed relations might not be able to be reconstructed back to the original relation.

3) checking some dependencies may require joining the instances of the decomposed relations.

Lossless-join: is a join of two smaller relations yields back the original relation.

Normalization

1nf: if it contains an atomic value and no candidate key that uniquely identifies the rest of the columns(direct or indirect).

2nf: it should be in 1nf and table should not contain any partial dependencies and it has transitivity.

3nf: it should be in 2nf and no transitivity

bcnf:it should be in 3nf and every attribute in the table should depend on the key.

4nf: it should be in bcnf and doesn’t have multi-valued dependencies.

5nf: if it doesn’t contain any join dependencies.

3nf synthesis algo:

1) Make right hand side atomic

2)Elimination of extra Attributes:  
1. Reduction rule 1: XY --> Z and X --> Z then Y is  
extra on left side.  
2. Reduction rule 2: XY --> Z and X --> Y then Y is  
extra we need only X --> Z

3) remove redundancy

4)partition of groups: one group should have the same left side and non fully fd should be in a separate group

5) merge of equivalent keys

Multi valued Dependency: it occurs when two attributes in a table are independent of each other but, both depend on a third attribute.