Normalization

Is a scientific or step by step formal process to arrive at the correct set of tables in a database.

Reduces redundancy

What is redundancy? Unnecessary repeatation of same data in multiple locations

Update Anamolies

What is update anamolies? If you repeat same data in many locations, we may update them inconsistently. Example, In attendance table, if we have trainer name, it may be updated differently in different rows.

Functional Dependency

It is based on a column depends on another column or whole entity.

For example, In Candidate table, phone number is based on candidate. But State is based on city, not on candidate.

Large table split into multiple tables, with relationship like primary key of one table is foreign key of another table, this is what we do when we do normalization. If we split tables, we do normalization. If we merge tables, we do de-normalization.

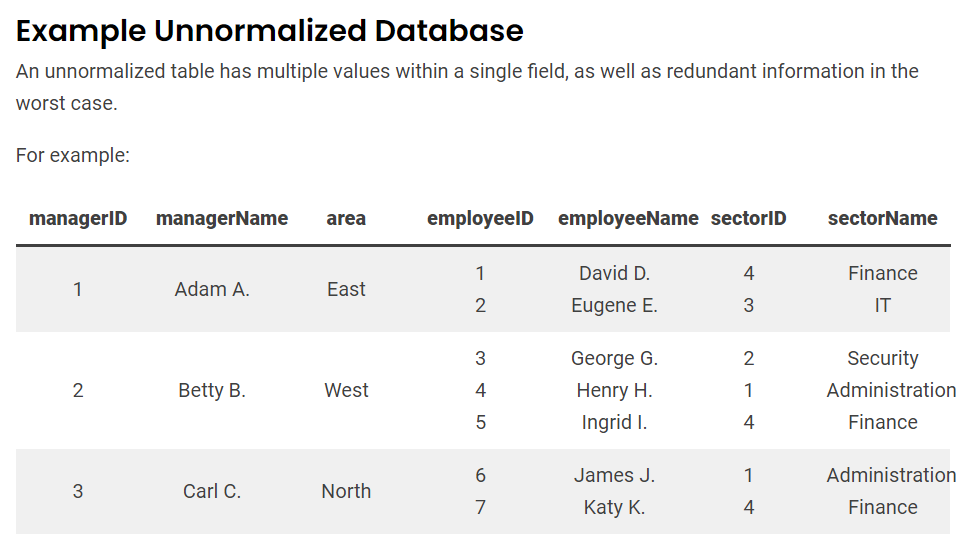
Step by Step normalization:

1st Normal Form

2nd Normal Form

3rd Normal Form

Boyce Codd Normal Form



First Normal Form:

A table is said to be in 1st Normal form if every cell in the table contains precisely only 1 data.

Ex:

SNO NAME ADDRESS

1 Raja 12, Main road, Chennai 600002

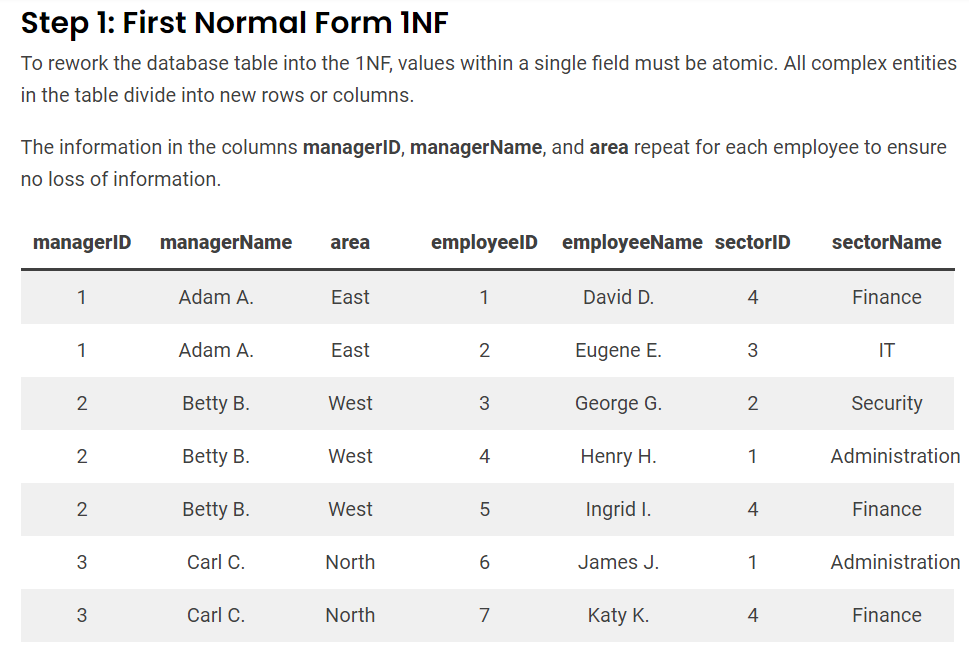
2 Siva 14, Raja street, Coimbatore 641002

The above table has a column address, which contains many data. So if it is like

SNO NAME® ADDRESS1 CITY PINCODE

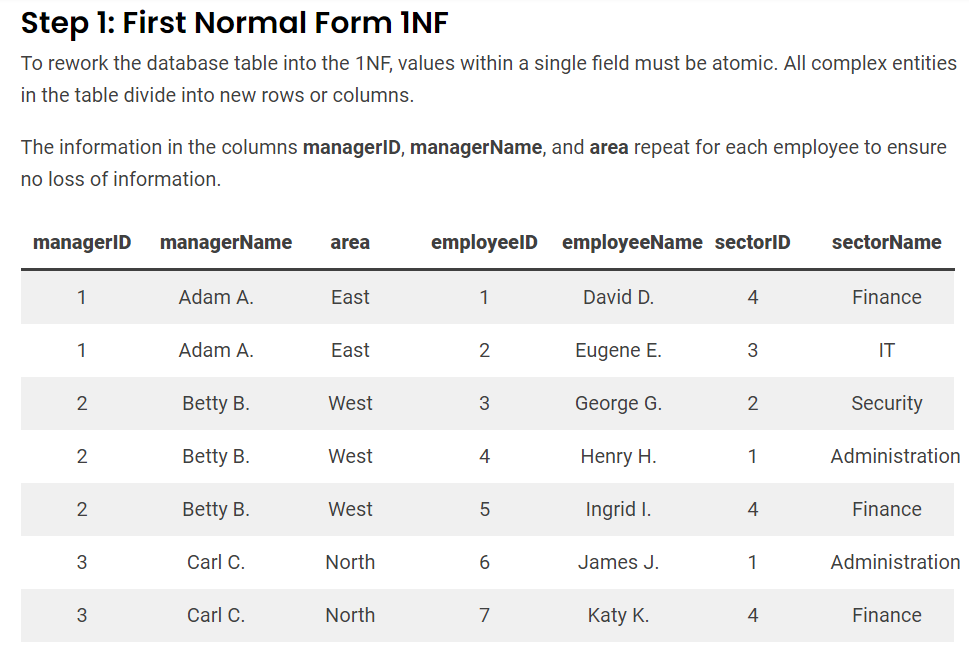
1 Raja 12, Main road, Chennai 600002

2 Siva 14, Raja street, Coimbatore 641002



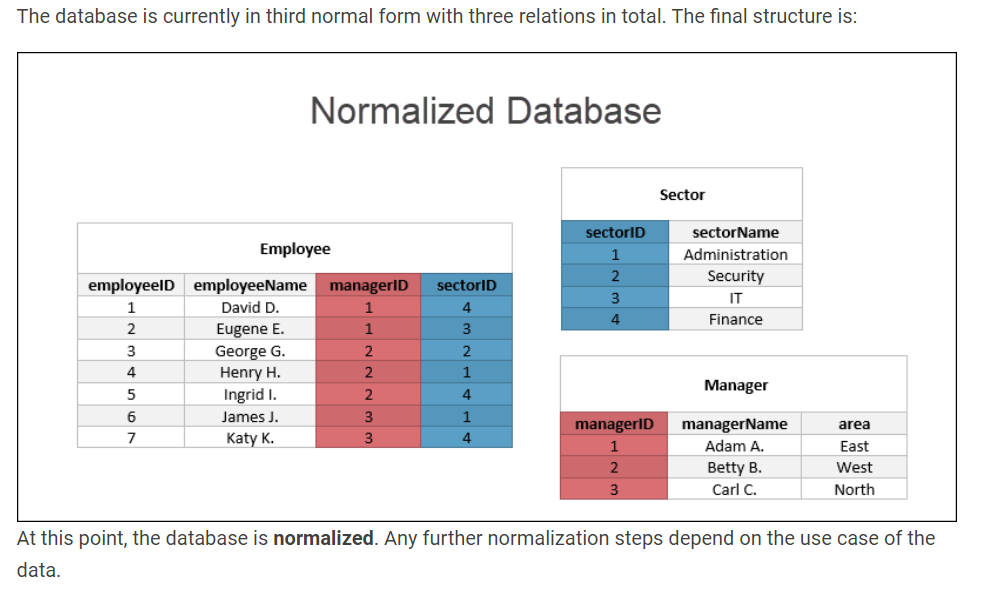
Second Normal Form:

A table is said to be in 2nd Normal form, if it is already in 1st normal form and every field in that table is dependent on whole key (entity).



Third Normal Form:

A table is said to be in 3rd Normal form, if it is already in 2nd normal form and every field is dependent on the primary key.



Activity:

Identify

First, Second and Third normal forms

Design patterns

Creational

Structural

Behavioral

SOLID principles

Single Responsibility Principle

Open – Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

1. Single Responsibility Principle

A class should implement only one functionality.

1. Open – Close Principle

Open for Extension, Closed for Modification.

1. Liskov Substitution Principle

If a method has an argument of type A, then we can substitute that with B or C or D, provided B, C, D are sub classes of A

1. Interface Seggregation Principle

Instead of declaring an object of class, it is always good to declare ref var of an interface

List<Integer> marks=new ArrayList<Integer>(); //this is better than

ArrayList<Integer> marks=new ArrayList<Integer>();

1. Dependency Inversion Principle

class Car1

{

Sony sony;

}

class Car2

{

Jbl jbl;

}

The above car classes have tightly coupled audio systems.

class Car3

{  
 Audio audio; //Audio is an interface

}

interface Audio

{}

class Sony implements Audio

{}

class Jbl implements Audio

{}

Reminder:

Creational

Structural

Behavioural

Stored Procedures

What is PLSQL?

Procedural Language Extension to SQL

SQL Statements are executed by sqL Engine.

PLSQL statements are executed by plsql engine.

In PLSQL

We can declare variables

We can use programming constructs like

Conditions and Looping

Every PLSQL program has a block structure

[DECLARE]

BEGIN

[EXCEPTION]

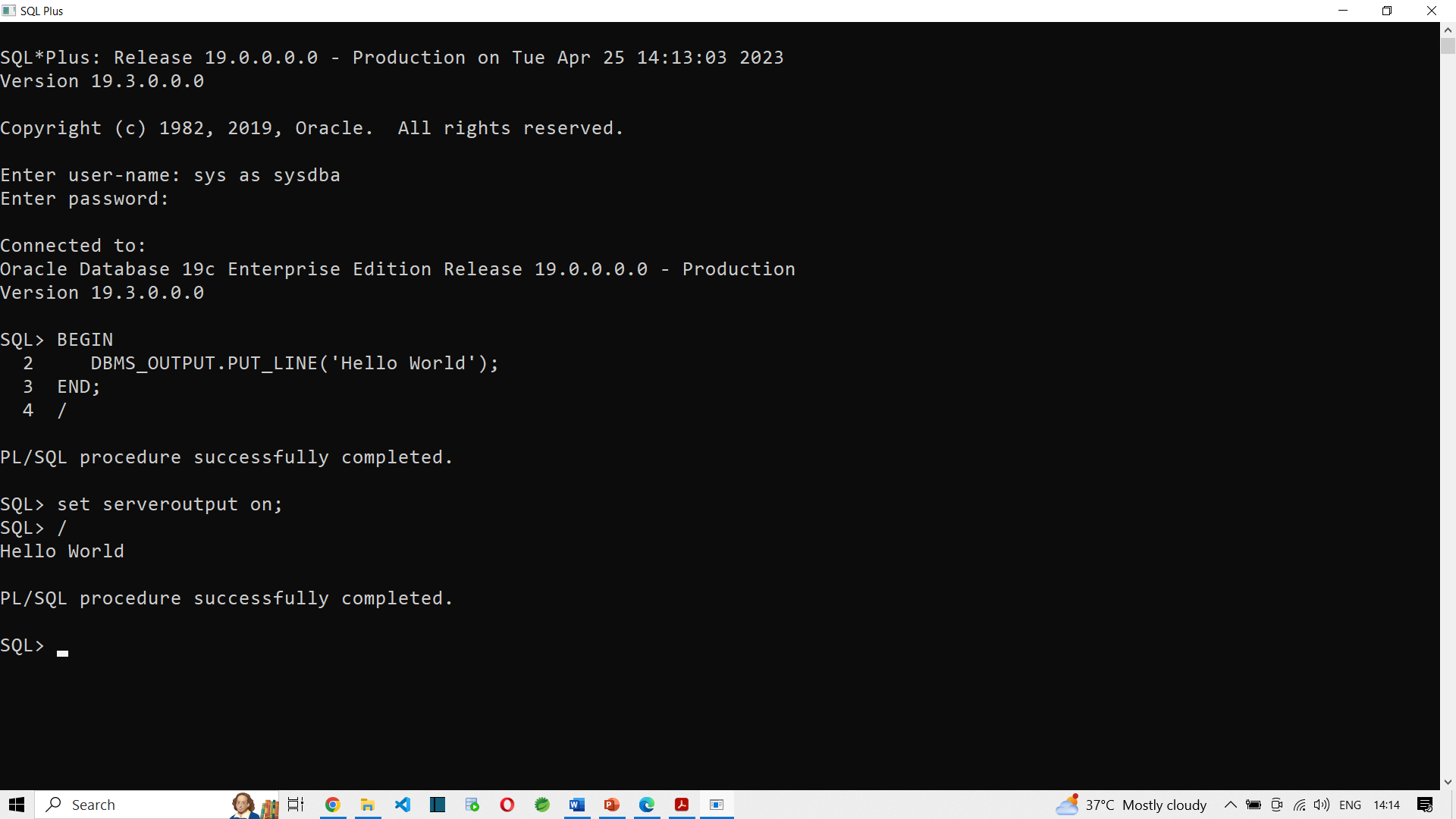
END

It is understood that a plsql program always have

BEGIN

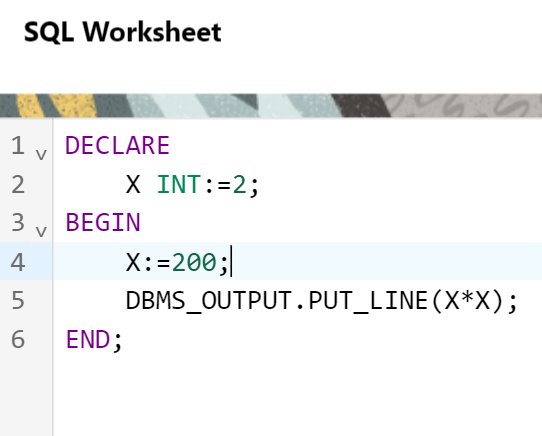
END;

set serveroutput on;



How to declare variables in PLSQL?

You can declare variables only in the DECLARE block



Can we run SQL statements inside PLSQL program?

Of course.

DECLARE

X INT;

BEGIN

SELECT 2\*3 INTO X FROM DUAL;

DBMS\_OUTPUT.PUT\_LINE(X);

END;

--

DECLARE

X INT;

BEGIN

SELECT SALARY INTO X FROM HR.EMPLOYEES; --More than 1 row is retrieved, so we cannot store it in a scalar variable

DBMS\_OUTPUT.PUT\_LINE(X);

END;

ORA-01422: exact fetch returns more than requested number of rows ORA-06512: at line 4

DECLARE

X INT;

BEGIN

SELECT SALARY INTO X FROM HR.EMPLOYEES WHERE EMPLOYEE\_ID=100;

DBMS\_OUTPUT.PUT\_LINE(X);

END;

Statement processed.  
24000

These are PLSQL programs. Also called as Anonymous programs.

Because these programs do not have a name. Hence they are not stored in database.

How to use IF conditions in PLSQL?

DECLARE

X INT;

BEGIN

SELECT SALARY INTO X FROM HR.EMPLOYEES WHERE EMPLOYEE\_ID=100;

IF X>20000 THEN

DBMS\_OUTPUT.PUT\_LINE('NOT ELIGIBLE FOR BONUS');

ELSIF X>10000 THEN

DBMS\_OUTPUT.PUT\_LINE('ELIGIBLE FOR 10% BONUS');

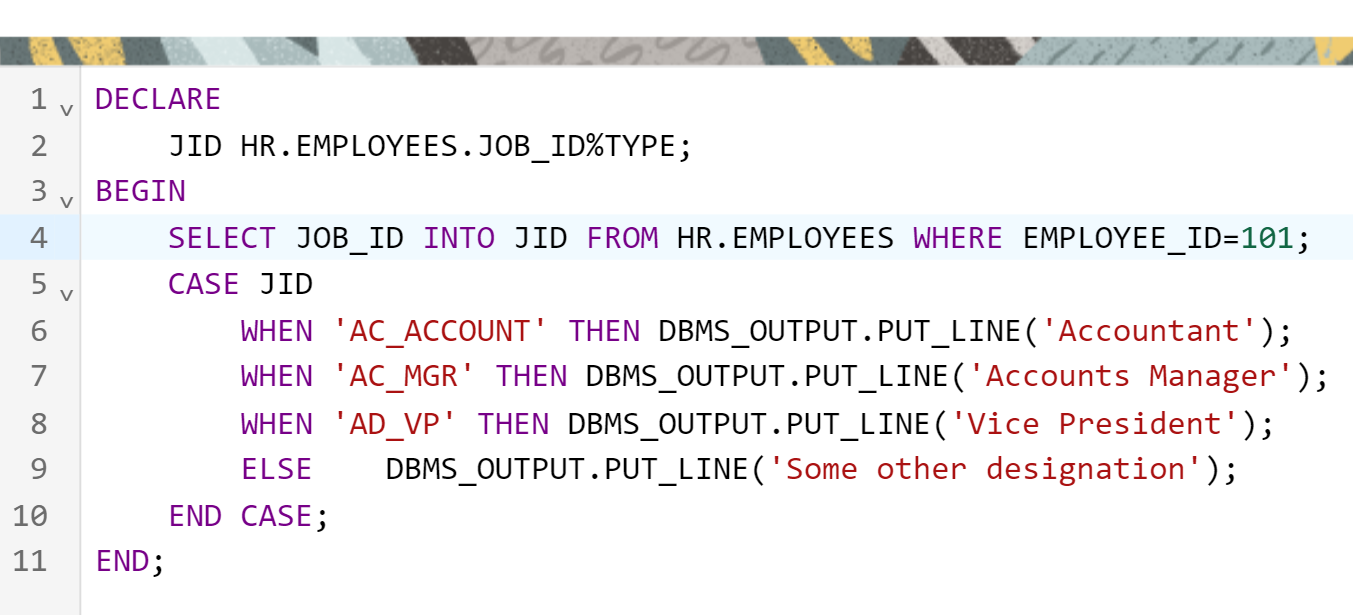
ELSE

DBMS\_OUTPUT.PUT\_LINE('ELIGIBLE FOR 50% BONUS');

END IF;

DBMS\_OUTPUT.PUT\_LINE(X);

END;



We can also do Looping:

DECLARE

JID HR.EMPLOYEES.JOB\_ID%TYPE;

I INT:=100;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE(I);

I:=I+1;

IF I>110 THEN

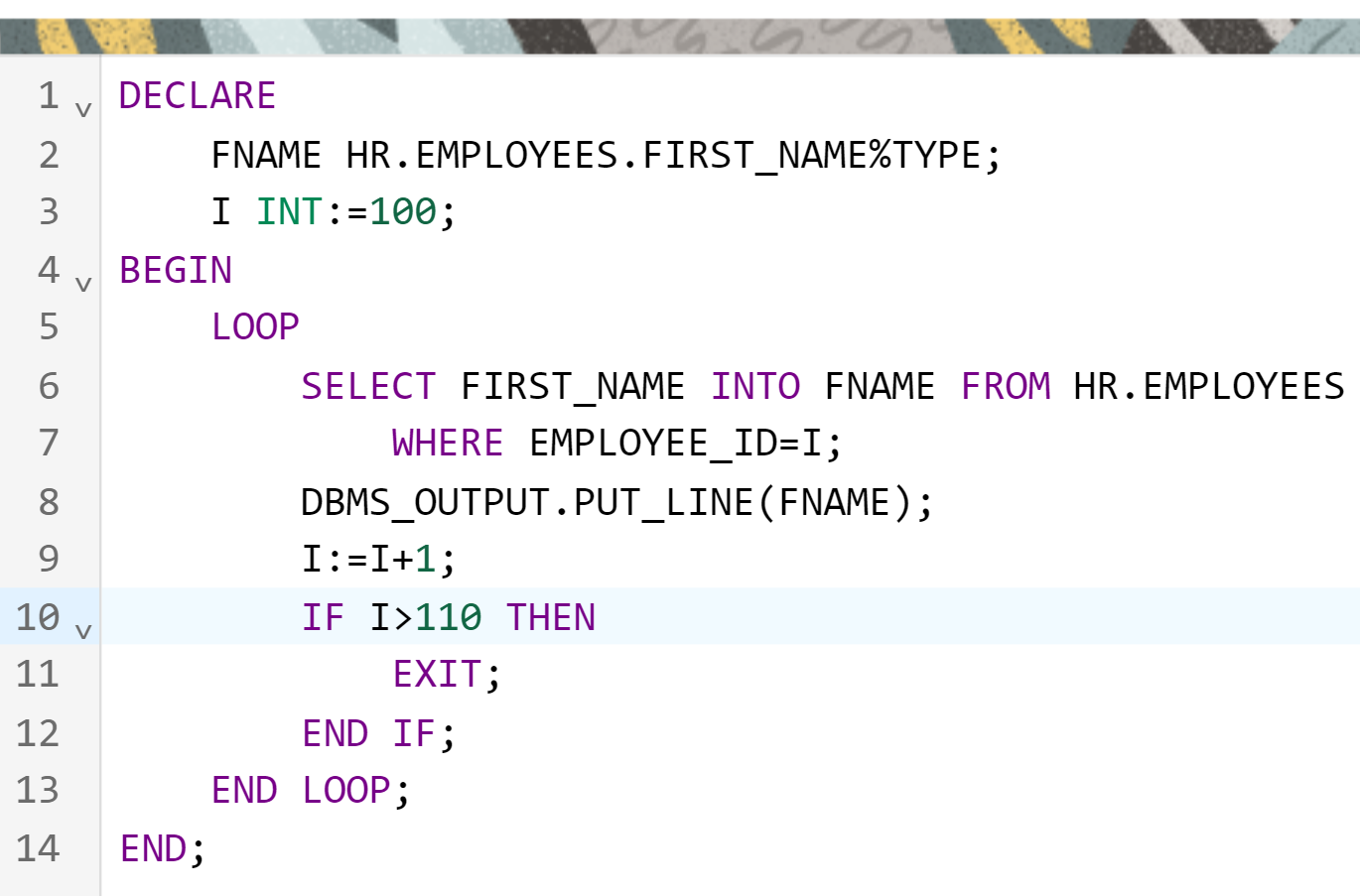
EXIT;

END IF;

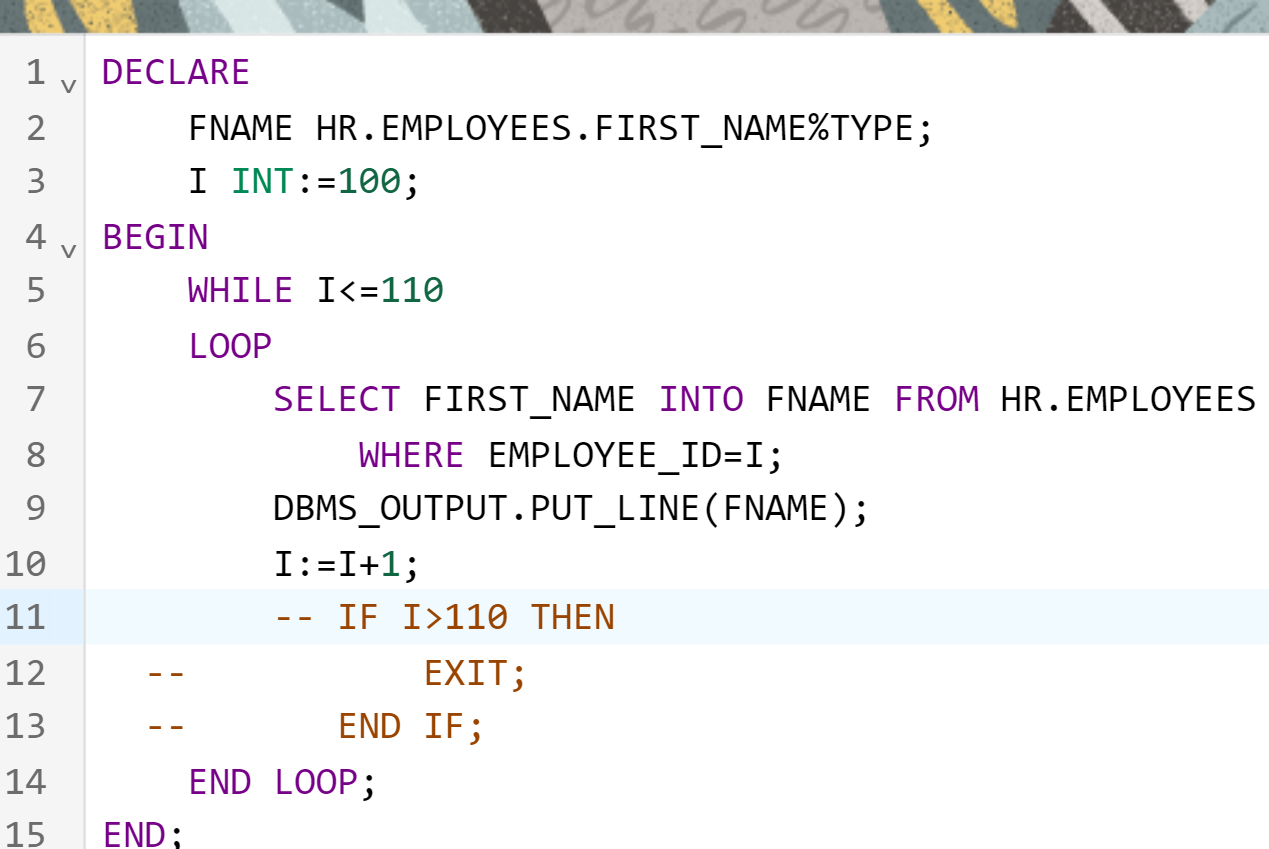
END LOOP;

END;

Statement processed.  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110

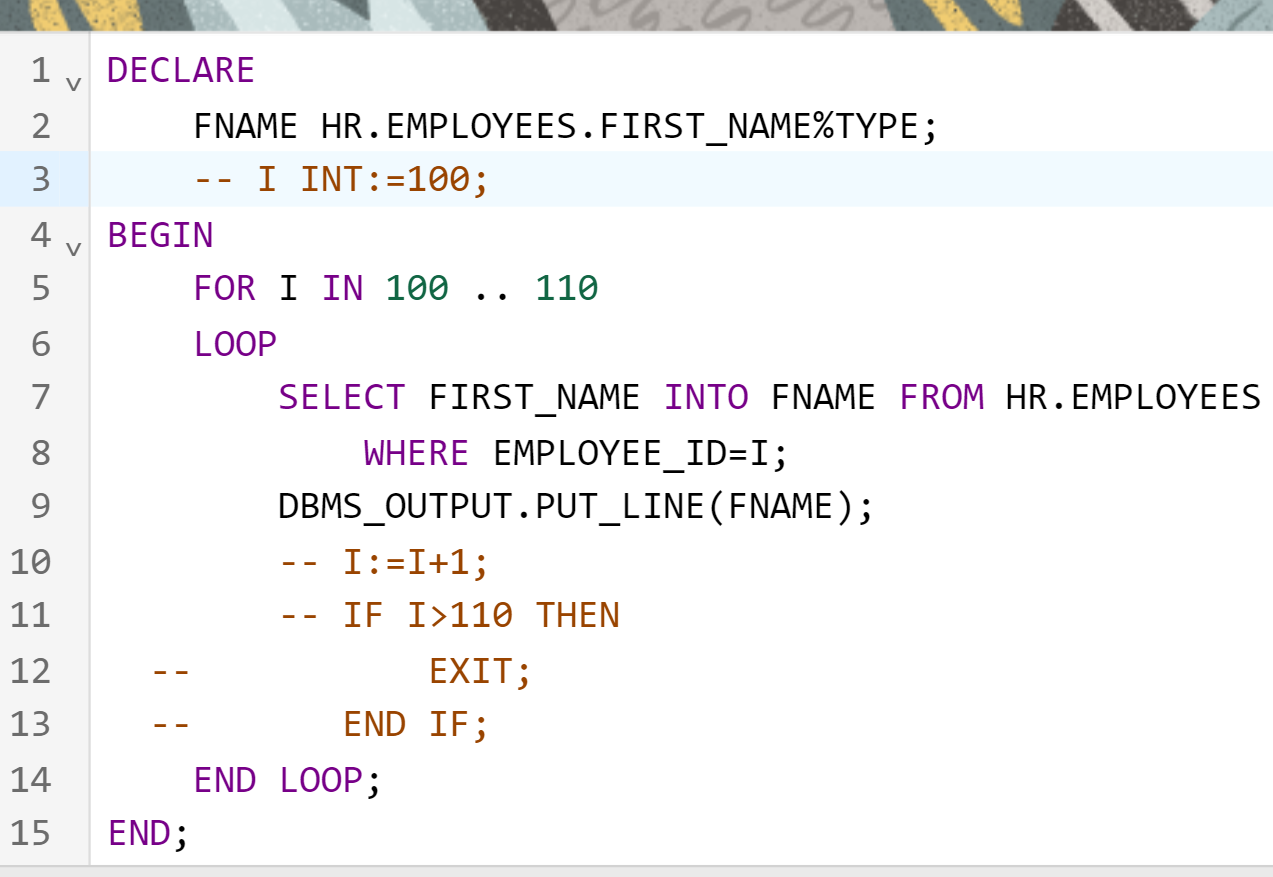


Statement processed.  
Steven  
Neena  
Lex  
Alexander  
Bruce  
David  
Valli  
Diana  
Nancy  
Daniel  
John

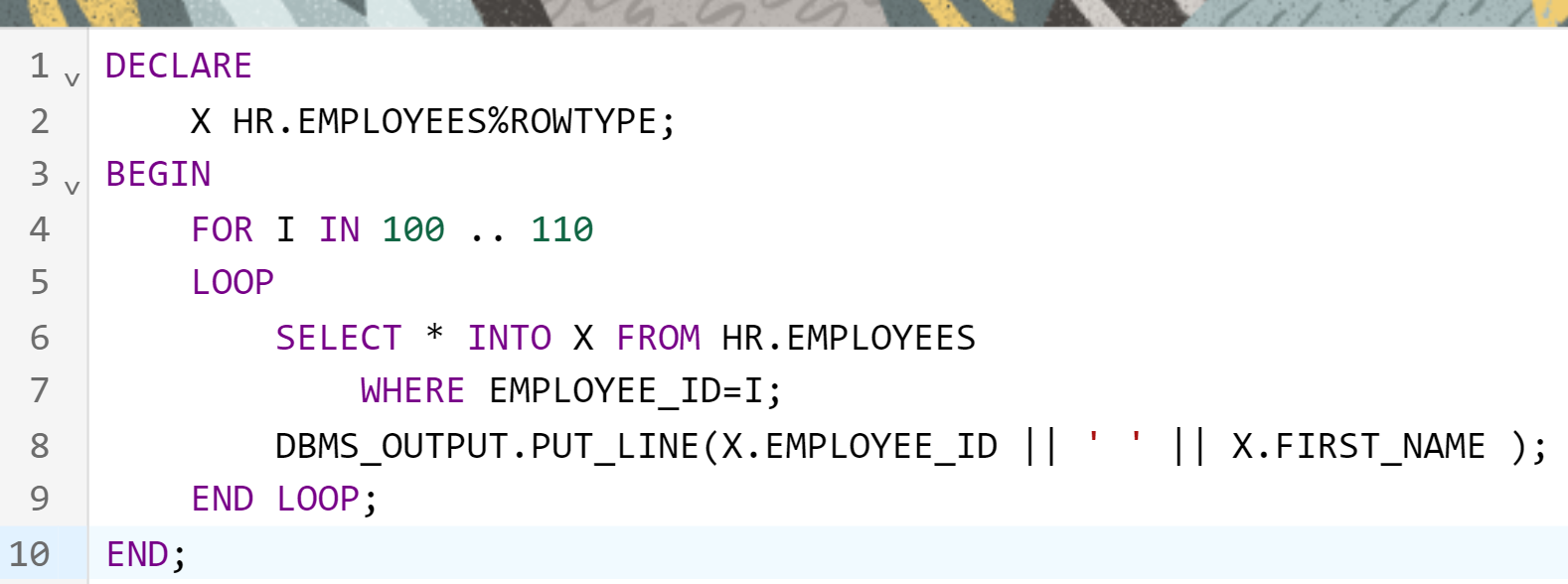


Same output. But we earlier used BASIC LOOP, now we use WHILE LOOP

Shall we use FOR LOOP now.



Notice that FOR loop variable need not be declared. If I is already declared, then it ignores that I and use Local loop variable I inside loop.



PROCEDURES

------------------------

They are named PLSQL programs

They are stored in the database as object.

We can call the procedure from

A plsql program (or from)

Another procedure (or from)

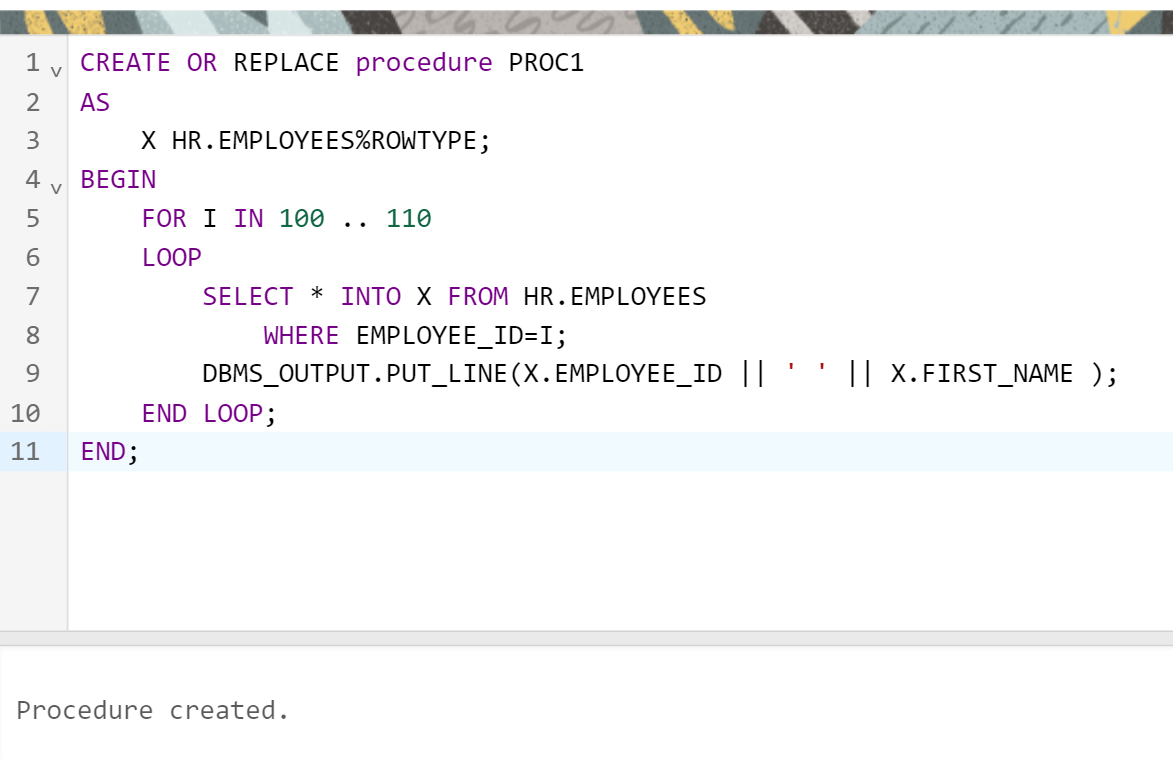
A function (or from)

A trigger

Etc

Procedure is like a void method in java.

I am going to modify the PLSQL program we did above into a procedure.



To call this procedure:

We can call this proc from a plsql program

BEGIN

proc1;

END;

Statement processed.  
100 Steven  
101 Neena  
102 Lex  
103 Alexander  
104 Bruce  
105 David  
106 Valli  
107 Diana  
108 Nancy  
109 Daniel  
110 John

PROCEDURES, can have parameters.

CREATE OR REPLACE procedure PROC1(M INT, N INT)

AS

X HR.EMPLOYEES%ROWTYPE;

BEGIN

FOR I IN M .. N

LOOP

SELECT \* INTO X FROM HR.EMPLOYEES

WHERE EMPLOYEE\_ID=I;

DBMS\_OUTPUT.PUT\_LINE(X.EMPLOYEE\_ID || ' ' || X.FIRST\_NAME );

END LOOP;

END;

To call this procedure:

DECLARE

X INT;

Y INT;

BEGIN

X:=100;

Y:=106;

PROC1(X,Y);

END;

Procedures can have

IN

OUT

IN OUT

Parameters.

Collections.sort(arr);

Sqr(2,x); x=4

Cube(2,x) x=8

Sqrt(4,x) x=2

Here, in above examples, x is OUT parameter, because the procedure fills output in that parameter.

CREATE OR REPLACE PROCEDURE PRCAJAY(X INT, Y OUT INT)

AS

BEGIN

Y:=X\*X;

END;

We call this procedure using PLSQL program:

DECLARE

X INT:=2;

Y INT;

BEGIN

PRCAJAY(2,Y);

DBMS\_OUTPUT.PUT\_LINE(Y);

END;

Output is 4

CREATE OR REPLACE PROCEDURE PRCSQR(X IN OUT INT)

AS

BEGIN

X:=X\*X;

END;

The above is IN OUT

DECLARE

X INT:=4;

BEGIN

PRCSQR(X);

DBMS\_OUTPUT.PUT\_LINE(X);

END;

Lets create a PLSQL Function.

CREATE OR REPLACE FUNCTION fnKalyan

RETURN INT

AS

BEGIN

RETURN 200;

END;

How to call the function?

A function can be called from

A plsql program

Another function

A procedure

SQL Statements

SELECT fnKalyan() FROM DUAL;

DECLARE

x INT;

BEGIN

x:=fnKalyan();

DBMS\_OUTPUT.PUT\_LINE(x);

END;

Demo:

Single row functions (scalar)

CREATE OR REPLACE FUNCTION fnBonus(SAL INT)

RETURN INT

AS

BEGIN

RETURN SAL+SAL\*2;

END;

Lets use this in our EMPLOYEES table

select first\_name, salary, fnBonus(salary) as Bonus

from hr.employees;

Task:

Create a plsql function that accepts employee id as input and returns the salary as output.

Task:

Create a plsql function that accepts employee id as input and returns the entire row as output.

create or replace function fn2(id1 INT)

return HR.EMPLOYEES%ROWTYPE

as

emprec HR.EMPLOYEES%ROWTYPE;

begin

SELECT \* INTO emprec FROM HR.EMPLOYEES WHERE EMPLOYEE\_ID=id1;

return emprec;

end;

DECLARE

result HR.EMPLOYEES%ROWTYPE;

BEGIN

result:=fn2(101);

DBMS\_OUTPUT.PUT\_LINE(result.FIRST\_NAME || ' ' || result.LAST\_NAME);

END;