***TALLER DE ADMINISTRACIÓN DE DATOS***

***TALLER 4 - PostgreSQL***

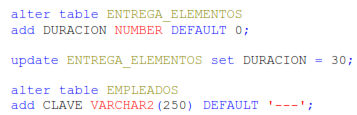
**INTEGRANTES:**

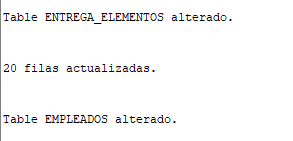
Johan Sneider Albarracin Gómez - 160003901

Edgar David Lozada González – 160004023

Carlos Danilo Núñez Gil – 160004032

Ejecute el siguiente código:

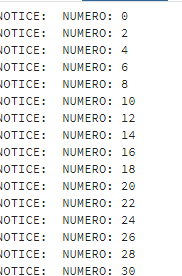




Realice los siguientes ejercicios:

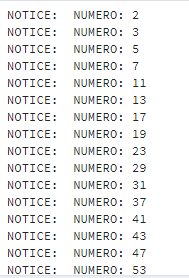
* Cree un bloque anónimo que visualice los números pares del 0 al 100.

1. **DO** $$
2. **DECLARE**
3. **BEGIN**
4. **FOR** numero **IN** 0..100 **LOOP**
5. **IF** **MOD**(numero, 2) = 0 **THEN**
6. RAISE NOTICE 'NUMERO: %',numero;
7. **END** **IF**;
8. **END** **LOOP**;
9. **END**;
10. $$ **LANGUAGE** PLPGSQL;



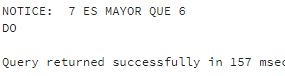
* Cree un bloque anónimo que visualice los números primos del 0 al 100.

1. **DO** $$
2. **DECLARE**
3. primo **BOOLEAN**;
4. **BEGIN**
5. **FOR** numero **IN** 0..100 **LOOP**
6. **FOR** divisor **IN** 2..numero **LOOP**
7. **IF** **MOD**(numero,divisor) = 0 **AND** numero!=divisor **THEN**
8. primo := **FALSE**;
9. EXIT;
10. **ELSE**
11. primo := **TRUE**;
12. **END** **IF**;
13. **END** **LOOP**;
14. **IF** primo = **TRUE** **THEN**
15. RAISE NOTICE 'NUMERO: %',numero;
16. **END** **IF**;
17. **END** **LOOP**;
18. **END**;
19. $$ **LANGUAGE** PLPGSQL;

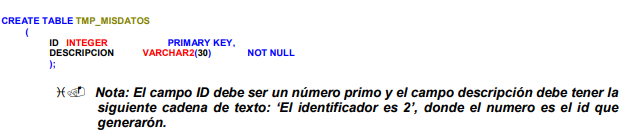


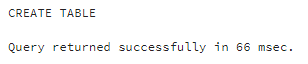
* Cree un bloque anónimo que solicite dos números y permita conocer cuál de los dos números es mayor o indicar si es el caso si son iguales.

1. **DO** $$
2. **DECLARE**
3. numero1 **INTEGER** := 7;
4. numero2 **INTEGER** := 6;
5. **BEGIN**
6. **IF** numero1 > numero2 **THEN**
7. RAISE NOTICE '% ES MAYOR QUE %',numero1,numero2;
8. ELSIF numero1 = numero2 **THEN**
9. RAISE NOTICE '% ES IGUAL QUE %',numero1,numero2;
10. **ELSE**
11. RAISE NOTICE '% ES MAYOR QUE %',numero2,numero1;
12. **END** **IF**;
13. **END**;
14. $$ **LANGUAGE** PLPGSQL;

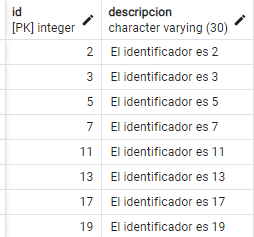


* Cree un bloque anónimo que ingrese 1000 registros a la siguiente tabla:



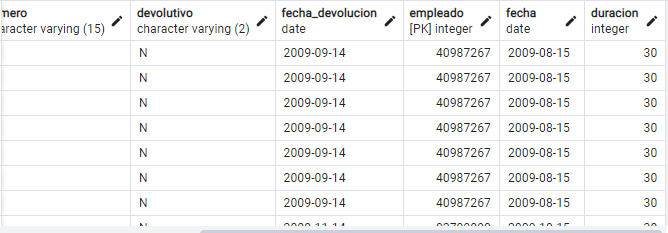


1. **DO** $$
2. **DECLARE**
3. primo **BOOLEAN**;
4. descrip VARCHAR(30) := 'El identificador es';
5. cantidad **INTEGER** := 1000;
6. contador **INTEGER** := 1;
7. numero **INTEGER** := 0;
8. **BEGIN**
9. **WHILE** contador <=cantidad **LOOP**
10. **FOR** divisor **IN** 2..numero **LOOP**
11. **IF** **MOD**(numero,divisor) = 0 **AND** numero!=divisor **THEN**
12. primo := **FALSE**;
13. EXIT;
14. **ELSE**
15. primo := **TRUE**;
16. **END** **IF**;
17. **END** **LOOP**;
18. **IF** primo = **TRUE** **THEN**
19. **INSERT** **INTO** tmp\_misdatos(id,descripcion) **VALUES** (numero,descrip||' '||numero);
20. contador := contador+1;
21. numero:= numero+1;
22. **ELSE**
23. numero := numero+1;
24. **END** **IF**;
25. **END** **LOOP**;
26. **COMMIT**;
27. **END**;
28. $$ **LANGUAGE** PLPGSQL;



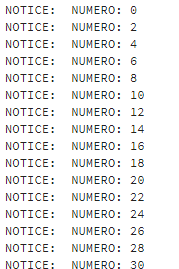
* Cree un bloque anónimo que actualice la fecha de devolución de la tabla ENTREGA\_ELEMENTOS, donde la duración (días) de cada elemento esta en el campo duración.

1. **DO** $$
2. **BEGIN**
3. **UPDATE** entrega\_elementos **SET** fecha\_devolucion=fecha+duracion;
4. **COMMIT**;
5. **END**;
6. $$ **LANGUAGE** PLPGSQL;



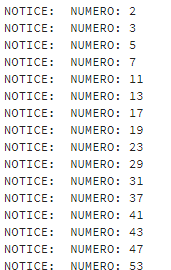
* Cree un procedimiento para el punto 1 con el nombre de: pares.

1. **CREATE** **OR** **REPLACE** **PROCEDURE** pares()
2. **AS** $$
3. **BEGIN**
4. **FOR** numero **IN** 0..100 **LOOP**
5. **IF** **MOD**(numero, 2) = 0 **THEN**
6. RAISE NOTICE 'NUMERO: %',numero;
7. **END** **IF**;
8. **END** **LOOP**;
9. **END**;
10. $$ **LANGUAGE** PLPGSQL;
11. CALL pares();



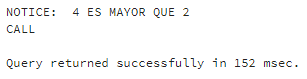
* Cree un procedimiento para el punto 2 con el nombre de: primos.

1. **CREATE** **OR** **REPLACE** **PROCEDURE** primos()
2. **AS** $$
3. **DECLARE**
4. primo **BOOLEAN**;
5. **BEGIN**
6. **FOR** numero **IN** 0..100 **LOOP**
7. **FOR** divisor **IN** 2..numero **LOOP**
8. **IF** **MOD**(numero,divisor) = 0 **AND** numero!=divisor **THEN**
9. primo := **FALSE**;
10. EXIT;
11. **ELSE**
12. primo := **TRUE**;
13. **END** **IF**;
14. **END** **LOOP**;
15. **IF** primo = **TRUE** **THEN**
16. RAISE NOTICE 'NUMERO: %',numero;
17. **END** **IF**;
18. **END** **LOOP**;
19. **END**;
20. $$ **LANGUAGE** PLPGSQL;
21. CALL primos();



* Cree un procedimiento para el punto 3 con el nombre de: comparar\_numeros, este procedimiento debe recibir por parámetros los dos números a comparar.

1. **CREATE** **OR** **REPLACE** **PROCEDURE** comparar\_numeros(numero1 **INTEGER**,numero2 **INTEGER**)
2. **AS** $$
3. **BEGIN**
4. **IF** numero1 > numero2 **THEN**
5. RAISE NOTICE '% ES MAYOR QUE %',numero1,numero2;
6. ELSIF numero1 = numero2 **THEN**
7. RAISE NOTICE '% ES IGUAL QUE %',numero1,numero2;
8. **ELSE**
9. RAISE NOTICE '% ES MAYOR QUE %',numero2,numero1;
10. **END** **IF**;
11. **END**;
12. $$ **LANGUAGE** plpgsql;
13. CALL comparar\_numeros(4,2)



* Cree un procedimiento para actualizar la columna email de la tabla empleados con la nueva dirección de correo asignada por la empresa. La nueva dirección asignada debe estar conformada por:

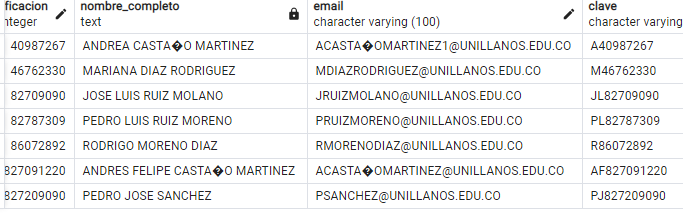
1. La inicial del primer nombre + el primer apellido + el segundo apellido + unillanos.edu.co. Ejemplo: Ricardo Andrés Sandoval Morales -> rsandovalmorales@unillanos.edu.co Además, debe generar la clave del correo la cual debe estar conformada por: Inicial del primer nombre + inicial del segundo nombre + la identificación.
2. Debe validar que no se repitan las direcciones, si se repita alguna, debe agregar un numero al final del nombre de la dirección de correo, el número es un consecutivo que debe iniciar en 1.

Función para buscar correos repetidos:

1. **CREATE** **OR** **REPLACE** **FUNCTION** buscar\_email\_repetido(correo empleados.email%**TYPE**,id empleados.identificacion%**TYPE**)
2. **RETURNS** **BOOLEAN** **AS** $$
3. **DECLARE**
4. contador **INTEGER** := 0;
5. **BEGIN**
6. **SELECT** **COUNT**(emp.email) **INTO** contador **FROM** empleados **AS** emp **WHERE** emp.email = correo **AND** emp.identificacion != id;
7. **IF** contador > 0 **THEN**
8. **RETURN** **TRUE**;
9. **ELSE**
10. **RETURN** **FALSE**;
11. **END** **IF**;
12. **END**;
13. $$ **LANGUAGE** plpgsql;

Procedimiento para ingresar la nueva dirección de correo:

1. **CREATE** **OR** **REPLACE** **PROCEDURE** actualizar\_correos()
2. **AS** $$
3. **DECLARE**
4. cur\_empleados **CURSOR** **FOR** **SELECT** nombre\_1, nombre\_2, apellido\_1, apellido\_2, identificacion **FROM** empleados;
5. correo\_nuevo empleados.email%**TYPE**;
6. clave\_nueva empleados.clave%**TYPE**;
7. dominio VARCHAR(17) := '@UNILLANOS.EDU.CO';
8. contador **INTEGER**;
9. **BEGIN**
10. -- Creamos un cursor para recorrer la tabla de empleados
11. **FOR** empleado **IN** cur\_empleados **LOOP**
12. -- Generamos la nueva direccion de correo
13. correo\_nuevo := CONCAT(**SUBSTR**(empleado.nombre\_1,1,1),empleado.apellido\_1,empleado.apellido\_2,dominio);
14. -- Generamos la clave del correo
15. clave\_nueva := CONCAT(**SUBSTR**(empleado.nombre\_1,1,1),**SUBSTR**(empleado.nombre\_2,1,1),empleado.identificacion);
16. --Validamos que no se repita la direccion de correo
17. **WHILE** buscar\_email\_repetido(correo\_nuevo,empleado.identificacion) = **TRUE** **LOOP**
18. contador := 1;
19. correo\_nuevo := CONCAT(**SUBSTR**(empleado.nombre\_1,1,1),empleado.apellido\_1,empleado.apellido\_2,contador,dominio);
20. contador := contador + 1;
21. **END** **LOOP**;
22. -- Actualizamos la direccion de correo y la clave del correo en la tabla de empleados
23. **UPDATE** empleados **SET** email = correo\_nuevo, clave = clave\_nueva **WHERE** identificacion = empleado.identificacion;
24. **COMMIT**;
25. **END** **LOOP**;
26. **END**;
27. $$ **LANGUAGE** plpgsql;
28. CALL actualizar\_correos();



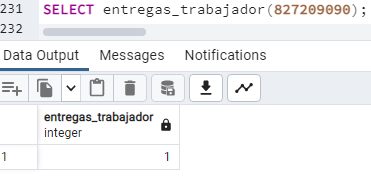
* Cree una función para conocer el precio promedio de un elemento de protección, la función debe recibir como parámetro el código del elemento.

1. **CREATE** **OR** **REPLACE** **FUNCTION** precio\_promedio\_elemento(cod\_elemento detalle\_entrada.elemento%**TYPE**)
2. **RETURNS** detalle\_entrada.v\_unitario%**TYPE** **AS** $$
3. **DECLARE**
4. precio\_promedio detalle\_entrada.v\_unitario%**TYPE**;
5. **BEGIN**
6. **SELECT** **AVG**(det.v\_unitario) **INTO** precio\_promedio **FROM** detalle\_entrada **AS** det **WHERE** det.elemento = cod\_elemento;
7. **RETURN** precio\_promedio;
8. **END**;
9. $$ **LANGUAGE** plpgsql;



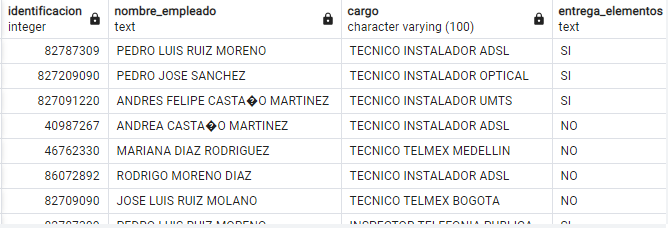
* Cree una función que determine si un trabajador actualmente se le han entregado todos los elementos de protección que le fueron asignados, la función debe retornar un valor entero(1 o 0).

1. **CREATE** **OR** **REPLACE** **FUNCTION** entregas\_trabajador(id\_empleado empleados.identificacion%**TYPE**)
2. **RETURNS** **INTEGER** **AS** $$
3. **DECLARE**
4. contador **INTEGER**;
5. **BEGIN**
6. **SELECT** **COUNT**(ea.empleado) **INTO** contador **FROM** elementos\_asignados **AS** ea **LEFT** **JOIN** entrega\_elementos **AS** ee **ON** ea.elemento = ee.elemento **WHERE** ee.id\_entrega **IS** **NULL** **AND** ea.empleado = id\_empleado;
7. **IF** contador >= 1 **THEN**
8. **RETURN** 0;
9. **ELSE**
10. **RETURN** 1;
11. **END** **IF**;
12. **END**;
13. $$ **LANGUAGE** plpgsql;



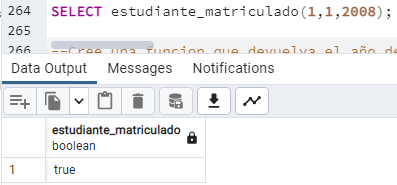
* Realizar una consulta que muestre: la identificación, el nombre del empleado, el cargo y una columna que indique si le entregaron (S/N) los elementos de protección.

1. **SELECT** emp.identificacion, CONCAT\_WS(' ',emp.nombre\_1,emp.nombre\_2,emp.apellido\_1,emp.apellido\_2) **AS** nombre\_empleado,
2. c.cargo,
3. **CASE**
4. **WHEN** entregas\_trabajador(emp.identificacion) = 1
5. **THEN** 'SI'
6. **ELSE** 'NO'
7. **END** **AS** entrega\_elementos
8. **FROM** empleados **AS** emp
9. **INNER** **JOIN** historial\_laboral **AS** hl **ON** emp.identificacion = hl.empleado
10. **INNER** **JOIN** cargos **AS** c **ON** hl.cargo = c.cod\_cargo;



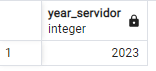
* Cree una función para conocer si un estudiante está matriculado actualmente, se entiende por matriculado que el estudiante por lo menos halla inscrito una materia en un periodo y año determinado. La función debe recibir como parámetro el código del estudiante, periodo y año, debe retornar un valor booleano.

1. **CREATE** **OR** **REPLACE** **FUNCTION** estudiante\_matriculado(cod\_est inscripciones.estudiante%**TYPE**, periodo\_ins inscripciones.periodo%**TYPE**, ano\_ins **INTEGER**)
2. **RETURNS** **BOOLEAN** **AS** $$
3. **DECLARE**
4. est\_matriculado **INTEGER**;
5. **BEGIN**
6. **SELECT** **COUNT**(\*) **INTO** est\_matriculado **FROM** inscripciones **WHERE** estudiante = cod\_est **AND** periodo = periodo\_ins **AND** ano = ano\_ins;
8. **IF** est\_matriculado >= 1 **THEN**
9. **RETURN** **TRUE**;
10. **ELSE**
11. **RETURN** **FALSE**;
12. **END** **IF**;
13. **END**;
14. $$ **LANGUAGE** plpgsql;



* Cree una función que devuelva el año de la fecha que tienen el servidor. La función no recibe parámetros, pero devuelve un valor numérico.

1. **CREATE** **OR** **REPLACE** **FUNCTION** year\_servidor()
2. **RETURNS** **INTEGER** **AS** $$
3. **BEGIN**
4. **RETURN** **EXTRACT**(**YEAR** **FROM** **NOW**());
5. **END**;
6. $$ **LANGUAGE** plpgsql;
7. **SELECT** year\_servidor();



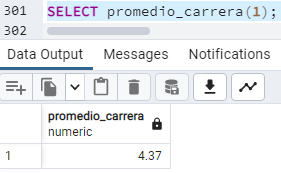
* Cree una función que determine el semestre del año en que se encuentra actualmente, tener como base la fecha del servidor. La función no recibe parámetros, pero devuelve un valor numérico.

1. **CREATE** **OR** **REPLACE** **FUNCTION** semestre\_servidor()
2. **RETURNS** **INTEGER** **AS** $$
3. **BEGIN**
4. **IF** **EXTRACT**(**MONTH** **FROM** **NOW**()) <=6 **THEN**
5. **RETURN** 1;
6. **ELSE**
7. **RETURN** 2;
8. **END** **IF**;
9. **END**;
10. $$ **LANGUAGE** plpgsql;

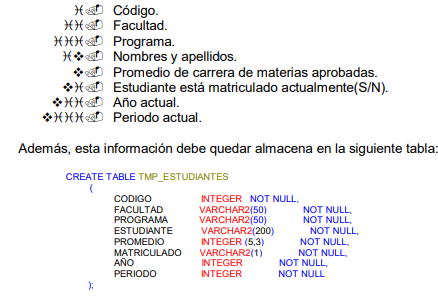


* Cree una función para conocer el promedio de carrera de un estudiante, solo tener en cuenta las materias aprobadas, la función debe recibir como parámetro el código del estudiante.

1. **CREATE** **OR** **REPLACE** **FUNCTION** promedio\_carrera(cod\_est inscripciones.estudiante%**TYPE**)
2. **RETURNS** inscripciones.nota%**TYPE** **AS** $$
3. **DECLARE**
4. promedio inscripciones.nota%**TYPE**;
5. **BEGIN**
6. **SELECT** **AVG**(ins.nota) **INTO** promedio **FROM** inscripciones **AS** ins **WHERE** ins.estudiante = cod\_est **AND** ins.nota >=3;
7. **RETURN** promedio;
8. **END**;
9. $$ **LANGUAGE** plpgsql;

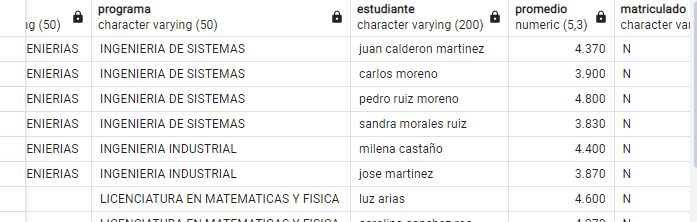


* Cree un procedimiento que muestre en pantalla la siguiente información de un estudiante:



Pero, solo para aquellos estudiantes cuyo promedio de carrera sea mayor o igual a 3.8. Cada vez que se llame el procedimiento, este debe vaciar la tabla TMP\_ESTUDIANTES.

1. **CREATE** **OR** **REPLACE** **PROCEDURE** informacion\_estudiantes()
2. **AS** $$
3. **DECLARE**
4. cur\_estudiantes **CURSOR** **FOR**
5. **SELECT** est.codigo **AS** codigo\_estudiante, fac.nombre **AS** facultad, pr.nombre **AS** programa,
6. CONCAT\_WS(' ',est.nombres,est.apellido1,est.apellido2) **AS** nombre\_completo **FROM** estudiante **AS** est
7. **INNER** **JOIN** programa **AS** pr **ON** est.facultad = pr.facultad **AND** est.programa = pr.id\_programa
8. **INNER** **JOIN** facultad **AS** fac **ON** est.facultad = fac.id\_facultad;
10. prom\_est tmp\_estudiantes.promedio%**TYPE**;
11. est\_matriculado VARCHAR(1);
12. ano\_actual **INTEGER**;
13. periodo\_actual **INTEGER**;
14. **BEGIN**
15. **DELETE** **FROM** tmp\_estudiantes;
16. ano\_actual := year\_servidor();
17. periodo\_actual := semestre\_servidor();
18. RAISE NOTICE 'informacion academica de los estudiantes';
19. **FOR** v\_estudiante **IN** cur\_estudiantes **LOOP**
20. prom\_est := promedio\_carrera(v\_estudiante.codigo\_estudiante);
21. **IF** estudiante\_matriculado(v\_estudiante.codigo\_estudiante,periodo\_actual,ano\_actual) = **TRUE** **THEN**
22. est\_matriculado := 'S';
23. **ELSE**
24. est\_matriculado := 'N';
25. **END** **IF**;
26. **IF** prom\_est >=3.8 **THEN**
27. RAISE NOTICE 'CODIGO---> %',v\_estudiante.codigo\_estudiante;
28. RAISE NOTICE 'FACULTAD---> %',v\_estudiante.facultad;
29. RAISE NOTICE 'PROGRAMA---> %',v\_estudiante.programa;
30. RAISE NOTICE 'NOMBRES Y APELLIDOS---> %',v\_estudiante.nombre\_completo;
31. RAISE NOTICE 'PROMEDIO DE CARRERA---> %',prom\_est;
32. RAISE NOTICE 'MATRICULADO---> %',est\_matriculado;
33. RAISE NOTICE 'AÑO ACTUAL---> %',ano\_actual;
34. RAISE NOTICE 'PERIODO ACTUAL---> %',periodo\_actual;
35. **INSERT** **INTO** tmp\_estudiantes(codigo,facultad,programa,estudiante,promedio,matriculado,ano,periodo)
36. VALUES(v\_estudiante.codigo\_estudiante,v\_estudiante.facultad,v\_estudiante.programa,v\_estudiante.nombre\_completo
37. ,prom\_est,est\_matriculado,ano\_actual,periodo\_actual);
38. **END** **IF**;
39. **END** **LOOP**;
40. **COMMIT**;
41. **END**;
42. $$ **LANGUAGE** plpgsql;
43. CALL informacion\_estudiantes();
44. **SELECT** \* **FROM** tmp\_estudiantes;



* Cree una función que retorne el número de estudiantes inscritos por materia, la función debe recibir como parámetros el código de la materia, el código del programa, código de facultad, el año y periodo.

1. **CREATE** **OR** **REPLACE** **FUNCTION** est\_inscritos\_materia(cod\_mat materias.id\_materia%**TYPE**, cod\_pro programa.id\_programa%**TYPE**, cod\_fac facultad.id\_facultad%**TYPE**, ano\_ins inscripciones.ano%**TYPE**, periodo\_ins inscripciones.periodo%**TYPE**)
2. **RETURNS** **INTEGER** **AS** $$
3. **DECLARE**
4. cantidad **INTEGER**;
5. **BEGIN**
6. **SELECT** **COUNT**(\*) **INTO** cantidad **FROM** inscripciones **AS** ins **WHERE** ins.materia = cod\_mat **AND**
7. ins.programa = cod\_pro **AND** ins.facultad = cod\_fac **AND** ins.ano = ano\_ins **AND** ins.periodo = periodo\_ins;
8. **RETURN** cantidad;
9. **END**;
10. $$ **LANGUAGE** plpgsql;
12. **SELECT** est\_inscritos\_materia(2,1,1,2008,1);



* Cree un dato compuesto de tipo tabla, que almacene la información del punto 18 (Solo la primera parte), con base en esta información debe mostrar:

1. El índice de la tabla y los datos.
2. Total de datos de la tabla.
3. Validar si la clave 10 existe, y si existe debe mostrar la información que contiene, si no, debe indicar que no existe.
4. Mostrar la información de la primera y la última clave.
5. Eliminar los índices de la tabla comprendidos entre 1 y 5, utilice la función delete (m, n).
6. **CREATE** **OR** **REPLACE** **PROCEDURE** informacion\_18()
7. **AS** $$
8. **DECLARE**
9. inscritos\_tabla inscripciones\_tabla[];
10. rm\_inscritos\_tabla inscripciones\_tabla[];
11. pos **INTEGER** := 0;
12. cur\_inscripciones **CURSOR** **FOR**
13. **SELECT** **ROW\_NUMBER**() OVER() **AS** indice, t.materia, t.cantidad FROM(**SELECT** nombre **AS** materia, **COUNT**(id\_materia) **AS** cantidad **FROM** inscripciones ins
14. **INNER** **JOIN** materias **ON** ins.materia = materias.id\_materia
15. **GROUP** **BY** nombre) **AS** t;
17. **BEGIN**
18. RAISE NOTICE '/////////////////////////////////////////////////////';
19. RAISE NOTICE '1. EL INDICE DE LA TABLA Y LOS DATOS:';
20. **FOR** v\_inscritos **IN** cur\_inscripciones **LOOP**
21. RAISE NOTICE '----------------------------------------------------';
22. inscritos\_tabla[v\_inscritos.indice]:= v\_inscritos;
23. RAISE NOTICE 'INDICE: %',inscritos\_tabla[v\_inscritos.indice].indice;
24. RAISE NOTICE 'MATERIA: %',inscritos\_tabla[v\_inscritos.indice].materia;
25. RAISE NOTICE 'CANTIDAD: %',inscritos\_tabla[v\_inscritos.indice].cantidad;
26. **END** **LOOP**;
27. RAISE NOTICE '/////////////////////////////////////////////////////';
28. RAISE NOTICE '2. TOTAL DE DATOS DE LA TABLA.';
29. RAISE NOTICE '----------------------------------------------------';
30. RAISE NOTICE 'TOTAL DE DATOS: %',**ARRAY\_LENGTH**(inscritos\_tabla,1);
31. RAISE NOTICE '/////////////////////////////////////////////////////';
32. RAISE NOTICE '3. VALIDAR SI LA CLAVE 10 EXISTE, Y SI EXISTE DEBE MOSTRAR LA INFORMACION QUE CONTIENE, SI NO, DEBE INDICAR QUE NO EXISTE.';
33. **IF** inscritos\_tabla[10] **IS** **NOT** **NULL** **THEN**
34. RAISE NOTICE '----------------------------------------------------';
35. RAISE NOTICE 'INDICE: %',inscritos\_tabla[10].indice;
36. RAISE NOTICE 'MATERIA: %',inscritos\_tabla[10].materia;
37. RAISE NOTICE 'CANTIDAD: %',inscritos\_tabla[10].cantidad;
38. **ELSE**
39. RAISE NOTICE '----------------------------------------------------';
40. RAISE NOTICE 'LA CLAVE 10 NO EXISTE';
41. **END** **IF**;
42. RAISE NOTICE '/////////////////////////////////////////////////////';
43. RAISE NOTICE '4. MOSTRAR LA INFORMACION DE LA PRIMERA Y LA ULTIMA CLAVE';
44. RAISE NOTICE '----------------------------------------------------';
45. RAISE NOTICE 'PRIMER INDICE: %',inscritos\_tabla[**ARRAY\_LOWER**(inscritos\_tabla,1)].indice;
46. RAISE NOTICE 'MATERIA: %',inscritos\_tabla[**ARRAY\_LOWER**(inscritos\_tabla,1)].materia;
47. RAISE NOTICE 'CANTIDAD: %',inscritos\_tabla[**ARRAY\_LOWER**(inscritos\_tabla,1)].cantidad;
48. RAISE NOTICE '----------------------------------------------------';
49. RAISE NOTICE 'ULTIMO INDICE: %',inscritos\_tabla[**ARRAY\_UPPER**(inscritos\_tabla,1)].indice;
50. RAISE NOTICE 'MATERIA: %',inscritos\_tabla[**ARRAY\_UPPER**(inscritos\_tabla,1)].materia;
51. RAISE NOTICE 'CANTIDAD: %',inscritos\_tabla[**ARRAY\_UPPER**(inscritos\_tabla,1)].cantidad;
52. RAISE NOTICE '/////////////////////////////////////////////////////';
53. RAISE NOTICE '5. ELIMINAR LOS INDICES DE LA TABLA COMPRENDIDOS ENTRE 1 Y 5, UTILICE LA FUNCION DELETE(M,N)';
54. RAISE NOTICE '----------------------------------------------------';
55. **FOR** i **IN** 1..**ARRAY\_LENGTH**(inscritos\_tabla, 1) **LOOP**
56. **IF** i > 5 **THEN**
57. pos := pos+1;
58. rm\_inscritos\_tabla[pos] := inscritos\_tabla[i];
59. **END** **IF**;
60. **END** **LOOP**;
61. RAISE NOTICE 'ELIMINADOS';
62. RAISE NOTICE '/////////////////////////////////////////////////////';
63. RAISE NOTICE 'DATOS ACTUALES';
64. RAISE NOTICE '----------------------------------------------------';
65. inscritos\_tabla := rm\_inscritos\_tabla;
66. **FOR** i **IN** 1..**ARRAY\_LENGTH**(inscritos\_tabla,1) **LOOP**
67. RAISE NOTICE 'INDICE: %',inscritos\_tabla[i].indice;
68. RAISE NOTICE 'MATERIA: %',inscritos\_tabla[i].materia;
69. RAISE NOTICE 'CANTIDAD: %',inscritos\_tabla[i].cantidad;
70. RAISE NOTICE '----------------------------------------------------';
71. **END** **LOOP**;
72. **END**;
73. $$ **LANGUAGE** plpgsql;
74. CALL informacion\_18();

