



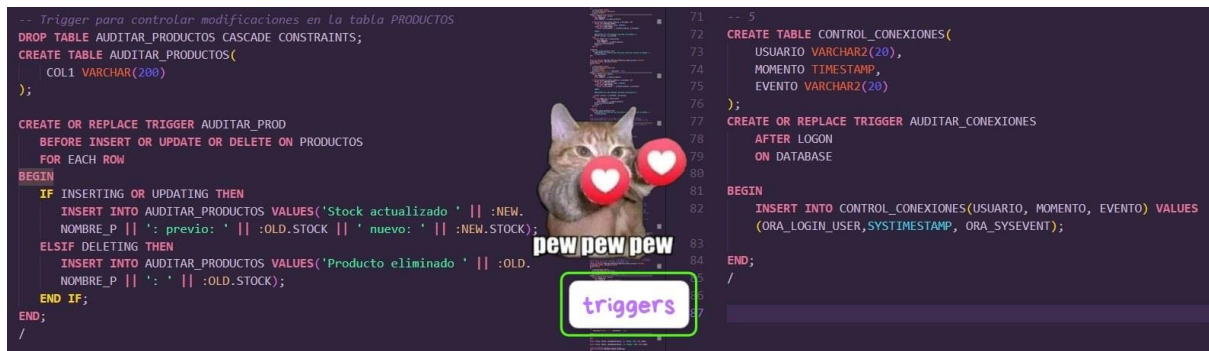
Proyecto Ampliación PL

He añadido a mi proyecto de bases de datos las siguientes funcionalidades haciendo uso de PL:

1. Función para verificar si hay stock.
2. Función para añadir productos a tu pedido. (Llama internamente a verificar stock).
3. Función anterior pero sobrecargada, aplicando descuento, y haciendo uso de un paquete de Oracle.
4. Función para clientes habituales con susodicha demostración.
5. Función para productos más vendidos.
6. Excepciones personalizadas en las funciones y procedimientos.
7. Procedimiento para eliminar productos impopulares.
8. Procedimiento para actualizar el stock y dependiendo del código aumentar más o menos.
9. Trigger para controlar el acceso a la base de datos.
10. Trigger para controlar la tabla productos.

Atentamente, Maki.

Triggers y añadir stock:



- **Añadir stock en productos, procedimientos, triggers y cursores:**
- Para añadir stock, he alterado la tabla Productos y añadido la columna stock. Pero está vacía para cada producto así que con un procedimiento con un cursor explícito voy a añadir por cada producto una cantidad. Para hacerlo más complejo, voy a traducir del código binario del código de producto, a decimal. Y según el resultado, añadiré 10 o 20 uds.
- Voy a introducir el trigger que guarda mis registros al modificar la tabla productos, porque quiero que estos primeros pasos se guarden en la tabla AUDITAR_PRODUCTOS

```

186/SQL> BEGIN
187/
188/  P1_NAME SQL.Statement ignored
189/  P2/SQL: ORA-00042: table or view does not exist
190/  P3/SQL: ORA-00042: table or view does not exist
191/  P4/SQL: ORA-00042: table or view does not exist
192/  CREATE OR REPLACE TRIGGER AUDITOR_PROD
193/  BEFORE INSERT OR UPDATE OR DELETE ON PRODUCTS
194/  FOR EACH ROW
195/  BEGIN
196/    IF INSERTING OR UPDATING THEN
197/      INSERT INTO AUDITOR_PROD VALUES('Stock actualizado' || :NEW.NOMBRE_P || ': ' || :NEW.STOCK || ' nuevo'; :NEW.STOCK)
198/    ELSE
199/      INSERT INTO AUDITOR_PROD VALUES('Producto eliminado' || :OLD.NOMBRE_P || ': ' || :OLD.STOCK);
200/    END IF;
201/  END;
202/
203/
204/Trigger created.

```

1 - Trigger que controla la tabla productos.

```

SQL> CREATE OR REPLACE TRIGGER BEFORE_AUDIT$CONTROL_CONEXIONS
1   AFTER LOGIN
2   ON DATABASE
3
4   BEGIN
5
6     INSERT INTO CONTROL_CONEXIONS(COLUARIO, MOMENTO, EVENTO) VALUES(COA_USER,SYSDATE$AMP,ORA_SYSTIME$TP);
7
8   END;
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
10
```

2 - Trigger que controla los logs de la base de datos.

Estos triggers se activarán automáticamente cada vez que se realicen modificaciones en las tablas PRODUCTOS y CLIENTES, lo que garantizará la integridad de la base de datos en tiempo real.

```

SQL> CREATE OR REPLACE PROCEDURE RellenarStock IS
2  CURSOR c_productos IS
3  SELECT CODIGO FROM PRODUCTOS;
4  v_codigo_producto_binario PRODUCTOS.CODIGO%TYPE;
5  v_codigo_producto_decimal NUMBER;
6  v_stock_actual PRODUCTOS.STOCK%TYPE;
7  codigo_incorrecto EXCEPTION;
8  BEGIN
9  OPEN c_productos;
10  FETCH c_productos INTO v_codigo_producto_binario;
11
12  WHILE c_productos%FOUND LOOP
13  v_codigo_producto_decimal := TO_NUMBER(v_codigo_producto_binario, 'B');
14  SELECT STOCK INTO v_stock_actual
15  FROM PRODUCTOS
16  WHERE CODIGO = v_codigo_producto_decimal;
17
18  IF v_codigo_producto_decimal > 255 THEN
19  RAISE codigo_incorrecto; --8 bits encendidos son 255
20  ELSE IF v_codigo_producto_decimal > 8 THEN
21  UPDATE PRODUCTOS
22  SET STOCK = v_stock_actual + 20
23  WHERE CODIGO = v_codigo_producto_decimal;
24  ELSE
25  UPDATE PRODUCTOS
26  SET STOCK = v_stock_actual + 10
27  WHERE CODIGO = v_codigo_producto_decimal;
28  END IF;
29  COMMIT;
30  FETCH c_productos INTO v_codigo_producto_binario;
31  END LOOP;
32  CLOSE c_productos;
33
34  DBMS_OUTPUT.PUT_LINE('Stock actualizado exitosamente.');
```

3 - Procedimiento para añadir stock.

Actualmente con **ALTER TABLE PRODUCTOS ADD STOCK NUMBER(10);** he añadido la columna STOCK, pero está vacía.

Lo del binario muy bonito pero un poco complicado, mi solución la he creado gracias a:

<https://web.archive.org/web/20150104004825/http://oraga.com/2008/02/29/how-to-convert-decimal-base-ten-number-to-binaryoctalhex-in-sql> y

<https://stackoverflow.com/questions/3086083/pl-sql-base-conversion-without-functions>

estos son los pasos que he seguido para realizar una función que me pase el binario a decimal:

1. Declaro 2 variables, para almacenar el número binario y otra variable para almacenar el resultado decimal.
2. Recorro el número binario de derecha a izquierda utilizando un bucle FOR inverso.
3. En cada iteración del bucle, obtengo el dígito binario correspondiente en la posición actual.
4. Calculo el valor decimal correspondiente al dígito binario actual utilizando la fórmula: $\text{valor_decimal} = \text{digito} * (2^{\text{posición}})$. La posición representa la posición del dígito binario en el número original, comenzando desde la posición 0.
5. Actualizo el valor decimal sumándole el cálculo realizado en el paso anterior.
6. Al finalizar el bucle, la variable decimal contendrá el valor decimal convertido del número binario original.

Como quiero hacerlo varias veces, lo he generado como función.

```

CREATE OR REPLACE FUNCTION BinarioADecimal(binario IN VARCHAR2) RETURN NUMBER IS
1  decimal NUMBER := 0;
2  BEGIN
3  FOR i IN REVERSE 1..LENGTH(binario) LOOP
4  DECLARE
5  digito NUMBER := TO_NUMBER(SUBSTR(binario, i, 1));
6  posicion NUMBER := LENGTH(binario) - i;
7  BEGIN
8  decimal := decimal + (digito * POWER(2, posicion));
9  END;
10 END LOOP;
11 RETURN decimal;
12 END;
13 /
```

4 - Código para pasar a decimal el código de producto.

```

SQL> DECLARE
2  CURSOR c1 IS
3  SELECT CODIGO FROM PRODUCTOS;
4  v_codigo_producto PRODUCTOS.CODIGO%TYPE;
5  BEGIN
6  OPEN c1;
7  LOOP
8  FETCH c1 INTO v_codigo_producto;
9  EXIT WHEN c1%NOTFOUND;
10 UPDATE PRODUCTOS SET STOCK = 15 WHERE CODIGO = v_codigo_producto;
11 COMMIT;
12 END LOOP;
13 CLOSE c1;
14 DBMS_OUTPUT.PUT_LINE('Valores insertados en la columna STOCK.');
```

Valores insertados en la columna STOCK.

PL/SQL procedure successfully completed.

```

SQL> select * from productos;
```

CODIGO	NOMBRE_P	PRECIO CIF	STOCK
000000001	PROGESTERONA	1 1234560M3	15
000000010	ESTRADIOL	2 1234560M3	15
000000100	DEPAKINE	3 1234560M3	15
000000011	HEURA	10 1234560M3	15
000000110	LECHE AVENA	1 1234560M3	15
000000111	LECHE SOJA	1 1234560M3	15
001100000	TESTOGEL	2 1234560M3	15
000100001	MAKE UP	5 1234560M3	15
000100000	BINDER	12 1234560M3	15
000100111	BANDERA	4 1234560M3	15

10 rows selected.

5 - Inserto previamente todo a 15 con un cursor que va por cada fila poniendo el valor en la columna Stock. (Esto lo hago para mostrar el cambio con el otro procedimiento).

```

SQL> CREATE OR REPLACE PROCEDURE ActualizarStock IS
2  CURSOR c_productos IS
3  SELECT CODIGO FROM PRODUCTOS;
4  v_codigo_producto_binario PRODUCTOS.CODIGO%TYPE;
5  v_codigo_producto_decimal NUMBER;
6  v_stock_actual PRODUCTOS.STOCK%TYPE;
7  codigo_incorrecto EXCEPTION;
8  BEGIN
9  OPEN c_productos;
10 FETCH c_productos INTO v_codigo_producto_binario;
11
12 WHILE c_productos%FOUND LOOP
13   v_codigo_producto_decimal := BinarioADecimal(v_codigo_producto_binario);
14   SELECT STOCK INTO v_stock_actual
15   FROM PRODUCTOS
16   WHERE CODIGO = v_codigo_producto_binario;
17
18   IF v_codigo_producto_decimal > 255 THEN
19     RAISE codigo_incorrecto; -- 8 bits encendidos son 255
20   ELSIF v_codigo_producto_decimal > 8 THEN
21     UPDATE PRODUCTOS
22     SET STOCK = v_stock_actual + 20
23     WHERE CODIGO = v_codigo_producto_binario;
24   ELSE
25     UPDATE PRODUCTOS
26     SET STOCK = v_stock_actual + 10
27     WHERE CODIGO = v_codigo_producto_binario;
28   END IF;
29   FETCH c_productos INTO v_codigo_producto_binario;
30 END LOOP;
31 COMMIT;
32 CLOSE c_productos;
33
34 DBMS_OUTPUT.PUT_LINE('Stock actualizado exitosamente.');
```

EXCEPTION

```

36 WHEN codigo_incorrecto THEN
37   DBMS_OUTPUT.PUT_LINE('Error al actualizar stock: 8 bits a 1 es 255 max');
38 WHEN OTHERS THEN
39   DBMS_OUTPUT.PUT_LINE('Error al actualizar el stock: ' || SQLERRM);
40 END;
```

Procedure created.

```

SQL> Execute ActualizarStock();
Stock actualizado exitosamente.
```

PL/SQL procedure successfully completed.

```

SQL> select * from productos;
```

CODIGO	NOMBRE_P	PRECIO CIF	STOCK
000000001	PROGESTERONA	1 1234560M3	25
000000010	ESTRADIOL	2 1234560M3	25
000000100	DEPAKINE	3 1234560M3	25
000000011	HEURA	10 1234560M3	25
000000110	LECHE AVENA	1 1234560M3	25
000000111	LECHE SOJA	1 1234560M3	25
001100000	TESTOGEL	2 1234560M3	35
000100001	MAKE UP	5 1234560M3	35
000100000	BINDER	12 1234560M3	35
000100111	BANDERA	4 1234560M3	35

6 - Salida del Actualizar Stock, añadiendo 10 o 20 según su binario.

El código emplea un cursor para recorrer cada fila de la tabla productos, cambia en cada iteración del loop valor del código que tiene con el fetch, pasa este de binario a decimal con la función previamente creada, y cambia la variable v_stock que pasa a albergar el stock correspondiente a la fila donde estemos.

Luego compara el valor decimal, y si es mayor de 255 da error, si es mayor a 8 aumenta en 20 el stock, y si es menor en 10.

El commit lo hago fuera del loop para hacer solo uno.

Y añado un mensaje para saber que todo salió bien.

```
181 CREATE OR REPLACE PROCEDURE ActualizarStock IS
182 CURSOR c_productos IS
183 SELECT CODIGO FROM PRODUCTOS;
184 v_codigo_producto_binario PRODUCTOS.CODIGOTYPE;
185 v_codigo_producto_decimal NUMBER;
186 v_stock_actual PRODUCTOS.STOCKTYPE;
187 codigo_incorrecto EXCEPTION;
188 BEGIN
189 OPEN c_productos;
190 FETCH c_productos INTO v_codigo_producto_binario;
191 WHILE c_productos%FOUND LOOP
192 v_codigo_producto_decimal := BinarioADecimal(v_codigo_producto_binario);
193 SELECT STOCK INTO v_stock_actual
194 FROM PRODUCTOS
195 WHERE CODIGO = v_codigo_producto_binario;
196 IF v_codigo_producto_decimal > 255 THEN
197 RAISE codigo_incorrecto; -- 8 bits encendidos son 255
198 ELSIF v_codigo_producto_decimal > 0 THEN
199 UPDATE PRODUCTOS
200 SET STOCK = v_stock_actual + 20
201 WHERE CODIGO = v_codigo_producto_binario;
202 ELSE
203 UPDATE PRODUCTOS
204 SET STOCK = v_stock_actual + 10
205 WHERE CODIGO = v_codigo_producto_binario;
206 END IF;
207 FETCH c_productos INTO v_codigo_producto_binario;
208 END LOOP;
209 COMMIT;
210 CLOSE c_productos;
211 DBMS_OUTPUT.PUT_LINE('Stock actualizado exitosamente.');
```

7 - Procedimiento ActualizarStock.

Como curiosidad sobre los triggers, añadiré que estuve buscando durante una hora cómo solucionar un error donde el anterior procedimiento solo me aumentaba el primer producto. (El error estaba en que cambie de nombre a la variable que se comparaba con el codigo de la tabla de producto, y en vez de compararse con la variable en binario, lo hacía con la decimal, de la cual solo coincide el 01).

El caso es que pese a tener mil intentos no se ha registrado en el trigger, pues no se ha llegado hacer un update con éxito. Y las capturas así lo demuestran.

```
18 IF v_codigo_producto_decimal > 255 THEN
19 RAISE codigo_incorrecto; -- 8 bits encendidos son 255
20 ELSIF v_codigo_producto_decimal > 0 THEN
21 UPDATE PRODUCTOS
22 SET STOCK = v_stock_actual + 20
23 WHERE CODIGO = v_codigo_producto_binario;
24 ELSE
25 UPDATE PRODUCTOS
26 SET STOCK = v_stock_actual + 10
27 WHERE CODIGO = v_codigo_producto_binario;
28 END IF;
29 FETCH c_productos INTO v_codigo_producto_binario;
30 END LOOP;
31 COMMIT;
32 CLOSE c_productos;
33 DBMS_OUTPUT.PUT_LINE('Stock actualizado exitosamente.');
```



8 - El error.

```
SQL> select * from auditar_productos;

COL1
-----
Stock actualizado PROGESTERONA: previo: nuevo:
Stock actualizado PROGESTERONA: previo: nuevo: 15
Stock actualizado ESTRADIOL: previo: nuevo: 15
Stock actualizado DEPAKINE: previo: nuevo: 15
Stock actualizado HEURA: previo: nuevo: 15
Stock actualizado LECHE AVENA: previo: nuevo: 15
Stock actualizado LECHE SOJA: previo: nuevo: 15
Stock actualizado TESTOGEL: previo: nuevo: 15
Stock actualizado MAKE UP: previo: nuevo: 15
Stock actualizado BINDER: previo: nuevo: 15
Stock actualizado BANDERA: previo: nuevo: 15

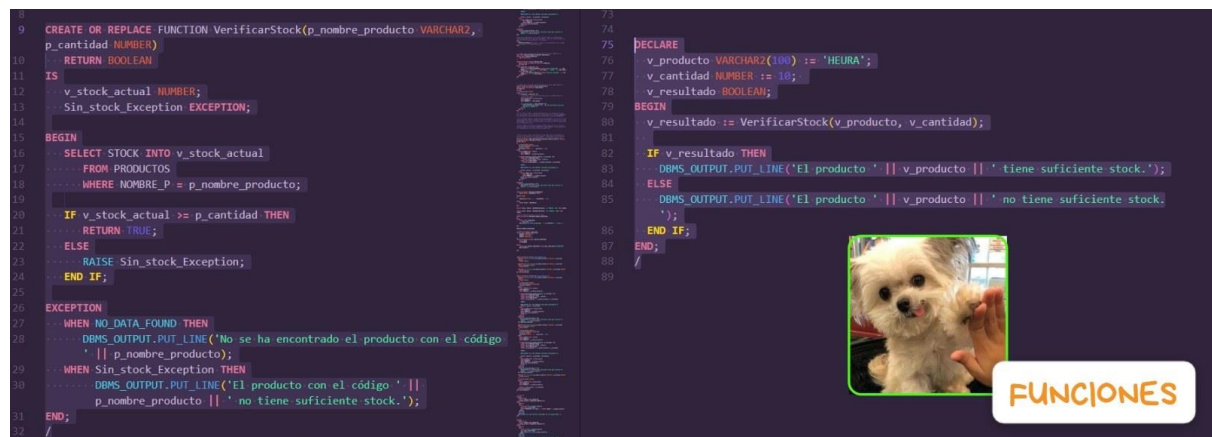
COL1
-----
Stock actualizado PROGESTERONA: previo: 15 nuevo: 25
Stock actualizado PROGESTERONA: previo: 25 nuevo: 35
Stock actualizado PROGESTERONA: previo: 35 nuevo: 45
Stock actualizado PROGESTERONA: previo: 45 nuevo: 55
Stock actualizado PROGESTERONA: previo: 55 nuevo: 65
Stock actualizado PROGESTERONA: previo: 65 nuevo: 15
Stock actualizado PROGESTERONA: previo: 15 nuevo: 25
Stock actualizado ESTRADIOL: previo: 15 nuevo: 25
Stock actualizado DEPAKINE: previo: 15 nuevo: 25
Stock actualizado HEURA: previo: 15 nuevo: 25
Stock actualizado LECHE AVENA: previo: 15 nuevo: 25

COL1
-----
Stock actualizado LECHE SOJA: previo: 15 nuevo: 25
Stock actualizado TESTOGEL: previo: 15 nuevo: 35
Stock actualizado MAKE UP: previo: 15 nuevo: 35
Stock actualizado BINDER: previo: 15 nuevo: 35
Stock actualizado BANDERA: previo: 15 nuevo: 35

27 rows selected.
```

9 - La tabla del trigger.

Funciones



```

9 CREATE OR REPLACE FUNCTION VerificarStock(p_nombre_producto VARCHAR2,
10 p_cantidad NUMBER)
11 RETURN BOOLEAN
12 IS
13     v_stock_actual NUMBER;
14     Sin_stock_Exception EXCEPTION;
15 BEGIN
16     SELECT STOCK INTO v_stock_actual
17     FROM PRODUCTOS
18     WHERE NOMBRE_P = p_nombre_producto;
19
20     IF v_stock_actual >= p_cantidad THEN
21         RETURN TRUE;
22     ELSE
23         RAISE Sin_stock_Exception;
24     END IF;
25
26 EXCEPTION
27 WHEN NO_DATA_FOUND THEN
28     DBMS_OUTPUT.PUT_LINE('No se ha encontrado el producto con el código
29     ' || p_nombre_producto);
30 WHEN Sin_stock_Exception THEN
31     DBMS_OUTPUT.PUT_LINE('El producto con el código ' ||
32     p_nombre_producto || ' no tiene suficiente stock.');
```

```

74
75 DECLARE
76     v_producto VARCHAR2(100) := 'HEURA';
77     v_cantidad NUMBER := 10;
78     v_resultado BOOLEAN;
79 BEGIN
80     v_resultado := VerificarStock(v_producto, v_cantidad);
81
82     IF v_resultado THEN
83         DBMS_OUTPUT.PUT_LINE('El producto ' || v_producto || ' tiene suficiente stock.');
```

FUNCIONES

Aquí empiezo por una función que verifica si hay suficiente stock, el parámetro es el nombre del producto y la cantidad.

Esta función la diseño para llamarla desde las funciones que realizan los pedidos, que serían aquellas que realizan mis clientes al darle a checkout de su carrito de la compra (si todo fuera bien pues les dejaría pagar, sino lanzaría un mensaje de error).

```

SQL> CREATE OR REPLACE FUNCTION VerificarStock(p_nombre_producto VARCHAR2, p_cantidad NUMBER)
2 RETURN BOOLEAN
3 IS
4     v_stock_actual NUMBER;
5     Sin_stock_Exception EXCEPTION;
6
7 BEGIN
8     SELECT STOCK INTO v_stock_actual
9     FROM PRODUCTOS
10     WHERE NOMBRE_P = p_nombre_producto;
11
12     IF v_stock_actual >= p_cantidad THEN
13         RETURN TRUE;
14     ELSE
15         RAISE Sin_stock_Exception;
16     END IF;
17
18 EXCEPTION
19 WHEN NO_DATA_FOUND THEN
20     DBMS_OUTPUT.PUT_LINE('No se ha encontrado el producto con el código ' || p_nombre_producto);
21 WHEN Sin_stock_Exception THEN
22     DBMS_OUTPUT.PUT_LINE('El producto con el código ' || p_nombre_producto || ' no tiene suficiente stock.');
```

```

23 END;
24 /
Function created.

SQL> DECLARE
2     v_producto VARCHAR2(100) := 'HEURA';
3     v_cantidad NUMBER := 10;
4     v_resultado BOOLEAN;
5 BEGIN
6     v_resultado := VerificarStock(v_producto, v_cantidad);
7
8     IF v_resultado THEN
9         DBMS_OUTPUT.PUT_LINE('El producto ' || v_producto || ' tiene suficiente stock.');
```

```

10 ELSE
11     DBMS_OUTPUT.PUT_LINE('El producto ' || v_producto || ' no tiene suficiente stock.');
```

```

12 END IF;
13 END;
14 /
El producto HEURA tiene suficiente stock.

PL/SQL procedure successfully completed.
```

10 - Verifica si hay suficiente stock.

A continuación, vamos a crear un procedimiento llamado *AddProductoPedido* que utilizará la función *VerificarStock* para verificar el stock antes de realizar un pedido. El procedimiento aceptará el código de producto y la cantidad solicitada como parámetros, y mostrará un mensaje de aviso si no hay suficiente stock.

Si todo va bien añadirá al pedido (a la línea de pedido) el producto y su cantidad. Además nos devolverá el precio.

Esta función está pensada para llamarse internamente y recursivamente desde un método o función *añadir pedido*. Que modifique la tabla cabecera_pedido. Es decir, *AddProductoPedido* se encarga de añadir nuevos productos a la línea de un producto ya existente. (He querido hacer mi código lo más modular posible).

```
IF VerificarStock(p_nombre_producto, p_cantidad) THEN
SELECT NVL(MAX(SUBSTR(NUM_LINEA, 2)), 0) + 1 INTO v_num_linea_number
FROM LINEA_PEDIDO
WHERE NUM_PEDIDO = p_num_pedido;
v_num_linea := 'P' || TO_CHAR(v_num_linea_number);
```

11 -

1. La idea es meter todo en la variable llamada *v_num_linea_number*.
2. Utilizo la función *NVL* para obtener el número máximo de línea (*NUM_LINEA*) de la tabla *LINEA_PEDIDO*. Si el resultado es nulo, se devolverá 0 (cero). El resultado se almacena en la variable *v_num_linea_number*.
3. Finalmente realizo una sentencia *SELECT* que obtendrá el número máximo (*NUM_LINEA*) de la tabla *LINEA_PEDIDO*. El resultado se almacenará en la variable *v_num_linea_number*.

Seguiré hablando de esta función al final del trabajo, junto con los paquetes y la sobrecarga.

Esta función nos devuelve el producto más vendido:

```
PRODUCTO_ESTRELLA()
-----
SQL> CREATE OR REPLACE FUNCTION producto_estrella
2 RETURN VARCHAR2
3 IS
4 v_codprod VARCHAR2(9);
5 v_nombreprod VARCHAR2(15);
6 BEGIN
7 SELECT COD_PRODUCTO INTO v_codprod
8 FROM (
9 SELECT COD_PRODUCTO, SUM(CANTIDAD) AS vendido
10 FROM LINEA_PEDIDO
11 GROUP BY COD_PRODUCTO
12 ORDER BY vendido DESC
13 ) WHERE ROWNUM = 1;
14
15 SELECT NOMBRE_P INTO v_nombreprod
16 FROM PRODUCTOS
17 WHERE CODIGO = v_codprod;
18
19 RETURN v_nombreprod;
20 END;
21 /
SQL> select producto_estrella() from dual;
PRODUCTO_ESTRELLA()
-----
ESTRAIDOL
SQL>
```

12 - Producto más vendido (reutilizado de ventas).

Esta nos devuelve un listado de clientes habituales, al principio me dirá que no ha encontrado a ninguno pues he añadido la condición de que hayan tenido que hacer más de 2 pedidos. Tras ello cambiaré mis tablas para que yo haya hecho un pedido más y por tanto aparezca como cliente habitual.

```

SQL> DROP procedure CLIENTES_HABITUALES;
SQL> CREATE OR REPLACE FUNCTION clientes_habituales
2 RETURN VARCHAR2
3 IS
4 v_result VARCHAR2(300);
5 v_message VARCHAR2(300);
6 m_clientes EXCEPTION;
7 BEGIN
8 SELECT LISTAGG('El cliente ' || nombre || ' con dni ' || dni || ' es habitual', ', ') WITHIN GROUP (ORDER BY dni)
9 INTO v_result
10 FROM cliente
11 WHERE dni IN (
12 SELECT dni
13 FROM cabecera_pedido
14 GROUP BY dni
15 HAVING COUNT(*) > 2
16 );
17 IF v_result IS NULL THEN
18 v_message := 'Sin resultados.';
19 RAISE m_clientes;
20 ELSE
21 v_message := 'Clientes habituales: ' || v_result;
22 END IF;
23 RETURN v_message;
24 EXCEPTION
25 WHEN no_clientes THEN
26 DBMS_OUTPUT.PUT_LINE('No se ha encontrado ningún cliente habitual.');
```

13 - Nos salta la excepción.

```

No se ha encontrado ningún cliente habitual.
SQL> INSERT INTO CABECERA_PEDIDO VALUES('P007','01/03/2023','511257377');
SQL> select clientes_habituales() from dual;

CLIENTES_HABITUALES()
-----
Clientes habituales: El cliente MAKI con dni 511257377 es habitual
SQL> =
```

14 - Tras añadir un pedido a mi nombre, aparezco ahora con 3 pedidos como cliente habitual.

Ahora bien, antes hicimos una función para ver el producto más vendido. Ordenando la lista al revés obtengo el menos vendido...

Y de ahí creo un procedimiento, que llama a esta función, para eliminar dicho producto.

```

SQL> CREATE OR REPLACE FUNCTION mostrar_producto_impopular
1 RETURN VARCHAR2
2 IS
3 v_codigo VARCHAR2(10);
4 v_nombre VARCHAR2(100);
5 BEGIN
6 SELECT prod_producto INTO v_codigo
7 FROM (
8 SELECT cod_producto, sum(cantvend) AS vendidas
9 FROM linea_pedido
10 GROUP BY cod_producto
11 ORDER BY vendidas asc
12 ) WHERE ROWNUM = 1;
13 SELECT nombre_p INTO v_nombre
14 FROM producto p
15 WHERE prod_producto = v_codigo;
16 RETURN v_codigo;
17 DBMS_OUTPUT.PUT_LINE('Si quieres puedes eliminar el producto ' || v_codigo || ' con el código ' || v_codigo || ' con el comando execute Eliminar_Producto_impopular(1);';
18 END;
19 SQL> select mostrar_producto_impopular() from dual;
20
21mostrar_producto_impopular()
-----
1000000000
```

15 - Mostrar producto menos vendido y sugerir su borrado.

```

SQL> CREATE OR REPLACE PROCEDURE Eliminar_Producto IS
2 v_codprod VARCHAR2(9);
3 v_nombprod VARCHAR2(15);
4 BEGIN --llama a mostrar producto impopular
5 DELETE FROM PRODUCTOS
6 WHERE CODIGO = mostrar_producto_impopular();
7 DBMS_OUTPUT.PUT_LINE('Eliminadas ' || SQL%ROWCOUNT || ' filas.');
```

16 - Eliminar producto menos vendido.

Todo esto era para demostrar algo básico de las bases de datos relacionales: va a producir errores, pues hay otras tablas, como línea de pedido que tienen valores de la fila que acabo de eliminar. (La famosa integridad referencial).


```

ERROR at line 1:
ORA-04091: table SYSTEM.LINEA_PEDIDO is mutating, trigger/function may not see it
ORA-06512: at "SYSTEM.MOSTRAR_PRODUCTO_POPULAR", line 7
ORA-06512: at "SYSTEM.ELIMINAR_PRODUCTO", line 5
ORA-06512: at line 1

SQL> select * from productos;

CODIGO Producto Precio unitario CIF STOCK
-----
0000000001 PROGESTERONA 1 123456000 25
0000000010 ESTRADOL 2 123456000 25
0000000100 DEPAKINE 3 123456000 25
0000000011 HEURA 10 123456000 25
0000000110 LECHE AVENA 1 123456000 25
0000000111 LECHE SOJA 1 123456000 25
0011000000 TESTOGEL 2 123456000 35
0001000001 MALE UP 5 123456000 35
0001000000 BINDER 12 123456000 35
0001000111 BANDERA 4 123456000 35

SQL> select * from linea_pedido;

ID Cantidad Numero de pedido COD_PROD
-----
P1 6 P001 0000000001
P2 7 P001 0000000011
P3 1 P001 0001000001
P2 1 P002 0000000110
P2 4 P002 0000000010
P1 5 P003 0000000010
P1 6 P004 0001000111
P1 7 P005 0000000100
P2 4 P005 0011000000
P3 1 P005 0000000111
P1 1 P006 0000000100
P2 1 P006 0001000000
SQL>

```

17 - Da un error y no se llega a borrar, gracias a esto se evita la catástrofe =)

```

-- Trigger para controlar modificaciones en la tabla LINEA_PEDIDO 809
CREATE OR REPLACE TRIGGER VerificarStockPedido
BEFORE INSERT OR UPDATE ON LINEA_PEDIDO
FOR EACH ROW
DECLARE
    v_stock_actual NUMBER;
BEGIN
    IF INSERTING OR UPDATING THEN
        SELECT STOCK INTO v_stock_actual
        FROM PRODUCTOS
        WHERE NOMBRE_P = :OLD.CODIGO;

        IF v_stock_actual < :NEW.CANTIDAD THEN
            RAISE_APPLICATION_ERROR(-20001, 'No hay suficiente stock para realizar el pedido.');
        END IF;
    END IF;
END;
/

```

18 - Trigger innecesario pero me gustaba la idea.

```

Run SQL Command Line

USUARIO          MOMENTO          EVENTO
-----
SYSTEM          28/05/23 00:23:51,340000 LOGON
SYSTEM          28/05/23 00:24:47,570000 LOGON
SYS             28/05/23 00:25:00,024000 LOGON
SYSTEM          28/05/23 00:25:57,043000 LOGON
SYS             28/05/23 00:30:00,014000 LOGON
SYS             28/05/23 00:30:01,025000 LOGON
SYS             28/05/23 00:35:00,020000 LOGON
SYS             28/05/23 00:40:00,023000 LOGON
SYS             28/05/23 00:45:00,030000 LOGON
SYS             28/05/23 00:50:00,015000 LOGON
SYSTEM          28/05/23 00:52:10,053000 LOGON
SYS             28/05/23 00:55:00,230000 LOGON
SYS             28/05/23 01:00:00,211000 LOGON
SYS             28/05/23 01:00:00,030000 LOGON
SYS             28/05/23 01:00:00,033000 LOGON
SYS             28/05/23 01:05:00,221000 LOGON
SYS             28/05/23 01:10:00,225000 LOGON
SYS             28/05/23 01:15:00,220000 LOGON
SYS             28/05/23 01:20:00,220000 LOGON
SYS             28/05/23 01:25:00,229000 LOGON
SYS             28/05/23 01:30:00,221000 LOGON
SYS             28/05/23 01:30:00,013000 LOGON
SYS             28/05/23 01:35:00,012000 LOGON
SYS             28/05/23 01:40:00,211000 LOGON
SYS             28/05/23 01:45:00,210000 LOGON
SYS             28/05/23 01:50:00,275000 LOGON
SYS             28/05/23 01:55:00,318000 LOGON
SYS             28/05/23 02:00:00,316000 LOGON
SYS             28/05/23 02:00:00,022000 LOGON
SYS             28/05/23 02:00:00,022000 LOGON
SYS             28/05/23 02:05:00,321000 LOGON
SYS             28/05/23 02:10:00,316000 LOGON
SYS             28/05/23 02:15:00,313000 LOGON
SYS             28/05/23 02:20:00,301000 LOGON
SYS             28/05/23 02:25:00,413000 LOGON
SYS             28/05/23 02:30:00,414000 LOGON
SYS             28/05/23 02:30:00,026000 LOGON
SYS             28/05/23 02:35:00,415000 LOGON
SYS             28/05/23 02:40:00,412000 LOGON
SYS             28/05/23 02:45:00,415000 LOGON
SYS             28/05/23 02:50:00,413000 LOGON
SYS             28/05/23 02:55:00,424000 LOGON
SYS             28/05/23 03:00:00,412000 LOGON
SYS             28/05/23 03:00:00,024000 LOGON
SYS             28/05/23 03:00:00,030000 LOGON
SYS             28/05/23 03:00:01,049000 LOGON
SYS             28/05/23 03:00:01,067000 LOGON
SYS             28/05/23 03:05:00,422000 LOGON
SYS             28/05/23 03:10:00,420000 LOGON
SYS             28/05/23 03:15:00,415000 LOGON
SYS             28/05/23 03:20:00,416000 LOGON
SYS             28/05/23 03:25:00,420000 LOGON
SYS             28/05/23 03:30:00,431000 LOGON
SYS             28/05/23 03:30:00,026000 LOGON
SYS             28/05/23 03:35:00,415000 LOGON
SYS             28/05/23 03:40:00,413000 LOGON
SYS             28/05/23 03:45:00,416000 LOGON
SYS             28/05/23 03:50:00,413000 LOGON
SYS             28/05/23 03:55:00,416000 LOGON
SYS             28/05/23 04:00:00,414000 LOGON
SYS             28/05/23 04:00:00,024000 LOGON

```

19 - El trigger que controla la conexión va llenándose a medida que se gestiona la base de datos, esto es muy útil para saber si alguien hiciera algo indebido.

USUARIO	MOMENTO	EVENTO
SYS	28/05/23 04:10:00,421000	LOGON
SYS	28/05/23 04:15:00,410000	LOGON
SYS	28/05/23 04:20:00,425000	LOGON
SYS	28/05/23 04:25:00,473000	LOGON
SYS	28/05/23 04:30:00,514000	LOGON
SYS	28/05/23 04:30:00,954000	LOGON
SYS	28/05/23 04:35:00,519000	LOGON
SYS	28/05/23 04:40:00,515000	LOGON
SYS	28/05/23 04:45:00,513000	LOGON
SYS	28/05/23 04:50:00,512000	LOGON
SYS	28/05/23 04:55:00,513000	LOGON
SYS	28/05/23 05:00:00,611000	LOGON
SYS	28/05/23 05:00:00,514000	LOGON
SYS	28/05/23 05:00:00,819000	LOGON
SYS	28/05/23 05:05:00,500000	LOGON
SYS	28/05/23 05:10:00,516000	LOGON
SYS	28/05/23 05:15:00,515000	LOGON
SYS	28/05/23 05:20:00,624000	LOGON
SYS	28/05/23 05:25:00,627000	LOGON
SYS	28/05/23 05:30:00,621000	LOGON
SYS	28/05/23 05:30:00,616000	LOGON
SYS	28/05/23 05:35:00,620000	LOGON
SYS	28/05/23 05:40:00,612000	LOGON
SYS	28/05/23 05:45:00,612000	LOGON
SYS	28/05/23 05:50:00,615000	LOGON
SYS	28/05/23 05:55:00,622000	LOGON
SYS	28/05/23 06:00:02,006000	LOGON
SYS	28/05/23 06:00:02,091000	LOGON
SYS	28/05/23 06:00:02,140000	LOGON
SYS	28/05/23 06:00:02,140000	LOGON
SYS	28/05/23 06:00:02,216000	LOGON
SYS	28/05/23 06:00:02,278000	LOGON
SYS	28/05/23 06:05:00,217000	LOGON
SYS	28/05/23 06:10:00,230000	LOGON
SYS	28/05/23 06:15:00,212000	LOGON
SYS	28/05/23 06:20:00,216000	LOGON
SYS	28/05/23 06:25:00,213000	LOGON
SYS	28/05/23 06:30:00,115000	LOGON
SYS	28/05/23 06:30:00,325000	LOGON
SYS	28/05/23 06:35:00,319000	LOGON
SYS	28/05/23 06:40:00,315000	LOGON
SYS	28/05/23 06:45:00,318000	LOGON
SYS	28/05/23 06:50:00,321000	LOGON
SYS	28/05/23 06:55:00,316000	LOGON
SYS	28/05/23 07:00:00,120000	LOGON
SYS	28/05/23 07:00:00,335000	LOGON
SYS	28/05/23 07:00:00,336000	LOGON
SYS	28/05/23 07:00:00,810000	LOGON

20 - Voy a votar con las ojeras más profundas que se haya visto...

```
CREATE OR REPLACE FUNCTION AddProductoPedido(p_num_pedido VARCHAR2, p_nombre_producto VARCHAR2, p_cantidad NUMBER)
RETURN NUMBER
IS
    v_stock_actual NUMBER;
    sin_stock_exception EXCEPTION;
    vPrecio NUMBER;
    v_num_linea VARCHAR2(4);
    v_num_linea_number NUMBER;
BEGIN
    SELECT PRECIO INTO vPrecio
    FROM PRODUCTOS
    WHERE NOMBRE_P = p_nombre_producto;

    IF VerificarStock(p_nombre_producto, p_cantidad) THEN
        SELECT NVL(MAX(SUBSTR(NUM_LINEA, 1)), 0) + 1 INTO v_num_linea_number
        FROM LINEA_PEDIDO
        WHERE NUM_PEDIDO = p_num_pedido;

        v_num_linea := 'P' || TO_CHAR(v_num_linea_number);

        INSERT INTO LINEA_PEDIDO (NUM_LINEA, CANTIDAD, NUM_PEDIDO, COD_PRODUCTO)
        VALUES (v_num_linea, p_cantidad, p_num_pedido, p_nombre_producto);

        UPDATE CABECERA_PEDIDO
        SET FECHA = SYSDATE
        WHERE NUM_PEDIDO = p_num_pedido;

        COMMIT;

        DBMS_OUTPUT.PUT_LINE('Pedido realizado exitosamente!');
        RETURN vPrecio * p_cantidad;
    ELSE
        SELECT STOCK INTO v_stock_actual
        FROM PRODUCTOS
        WHERE NOMBRE_P = p_nombre_producto;
        RAISE sin_stock_exception;
        RETURN 0;
    END IF;
END IF;
```

21 - Intentando descifrar la manera de sumarle 1 a la última línea de la tabla línea pedidos, pero es un varchar con una letra delante, así que substrings, tochar, tonumber etc...

Paquetes Oracle

```
CREATE [OR REPLACE] PACKAGE nombre_paquete
IS | AS
    declaraciones de variables, cursores...
    subprogramas en PLSQL
END nombre_paquete
```



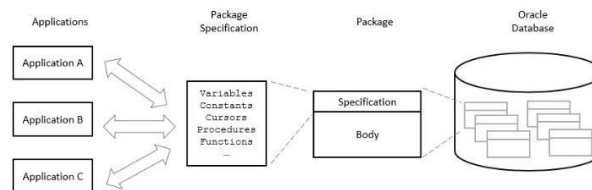


¿Qué es un paquete PL/SQL?

En PL/SQL, un paquete es un objeto de esquema que contiene definiciones para un grupo de funcionalidades relacionadas. Un paquete incluye variables, constantes, cursores, excepciones, procedimientos, funciones y subprogramas. Se compila y almacena en la base de datos de Oracle.

Por lo general, un paquete consta de una especificación y un cuerpo. La especificación del paquete es obligatoria, mientras que el cuerpo del paquete puede ser requerido u opcional, dependiendo de la especificación del paquete.

La siguiente imagen ilustra los paquetes PL/SQL:



Especificación del paquete La especificación del paquete declara los objetos públicos que son accesibles desde fuera del paquete.

Si una especificación del paquete incluye objetos públicos como cursores y subprogramas, entonces debe tener un cuerpo que defina las consultas para los cursores y el código para los subprogramas.

Cuerpo del paquete El cuerpo del paquete contiene la implementación de los cursores o subprogramas declarados en la especificación del paquete. En el cuerpo del paquete, puedes declarar o definir variables privadas, cursores, etc., utilizados solo por el cuerpo del paquete en sí mismo.

El cuerpo del paquete puede tener una parte de inicialización cuyas sentencias inicializan variables o realizan otras configuraciones únicas para todo el paquete.

El cuerpo del paquete también puede tener una parte de manejo de excepciones utilizada para manejar las excepciones.

Por qué usar paquetes PL/SQL El paquete es una característica poderosa de PL/SQL que debes utilizar en cualquier proyecto. A continuación, se presentan las ventajas del paquete:

Hacer el código más modular Los paquetes te permiten encapsular tipos, variables, constantes, subprogramas, cursores y excepciones lógicamente relacionados en módulos PL/SQL con nombre. Al hacer esto, haces que cada paquete sea más reutilizable, manejable, legible y confiable.

Ocultar detalles de implementación Los paquetes te permiten exponer la funcionalidad a través de sus especificaciones y ocultar la implementación detallada en el cuerpo del paquete.

Esto significa que puedes mejorar el código en el cuerpo del paquete sin afectar a otros paquetes o aplicaciones dependientes.

Mejorar el rendimiento de la aplicación Oracle carga el paquete en memoria la primera vez que invocas un subprograma del paquete. Las llamadas posteriores a otros subprogramas en el mismo paquete no requieren E/S de disco. Este mecanismo ayuda a mejorar el rendimiento.

Minimizar la recompilación innecesaria de código Los paquetes ayudan a evitar el proceso de recompilación innecesario. Por ejemplo, si cambias el cuerpo de una función del paquete, Oracle no recompila los subprogramas que usan la función, porque los subprogramas solo dependen de la especificación del paquete, no del cuerpo del paquete.

Gestionar la autorización fácilmente Al encapsular objetos en un paquete, concedes roles en el paquete en lugar de conceder roles en cada objeto del paquete.

El concepto de paquete PL/SQL es simple pero poderoso. Te permiten encapsular el código y hacer que tu aplicación sea más fácil de desarrollar y mantener.

El texto en Sway funciona como texto normal, pero con una vuelta de tuerca. Mientras escribe, el motor de diseño integrado de Sway combina el texto con otros contenidos multimedia para ofrecerle grandes resultados. Puede usar **Énfasis** o *Resaltado* para diferenciar la apariencia de las distintas partes del texto. El énfasis hace que el texto destaque notablemente. El resaltado es más sutil, pero proporciona un impacto visual similar. También puede agregar vínculos web, viñetas y listas numeradas.

Paquete con funciones sobrecargadas:



La sobrecarga implica que se llamen igual y devuelvan lo mismo, pero tengan diferentes parámetros. Así en tiempo de ejecución se decide a qué función se está llamando.

En una de mis funciones aplico un descuento como parámetro extra.

Y es bueno matizar, que depende si son funciones externas o internas al paquete se las invocaría diferente.

NombrePaquete.Función(Parámetros); si la función que invoca es externa.

Función(Parámetros); si es interna al paquete.

```
CREATE OR REPLACE PACKAGE DescuentoPedidoPackage AS
  FUNCTION CalcularDescuento(p_num_pedido VARCHAR2)
    RETURN NUMBER;

  FUNCTION CalcularDescuento(p_num_pedido VARCHAR2, p_porcentaje_descuento NUMBER)
    RETURN NUMBER;
END DescuentoPedidoPackage;
/

CREATE OR REPLACE PACKAGE BODY DescuentoPedidoPackage AS
  FUNCTION CalcularDescuento(p_num_pedido VARCHAR2)
    RETURN NUMBER IS
    v_coste_total NUMBER;
  BEGIN
    -- cálculo del coste total del pedido
    -- Aquí va el código para calcular el coste total del pedido con el número de pedido p_num_pedido

    RETURN v_coste_total;
  END CalcularDescuento;

  FUNCTION CalcularDescuento(p_num_pedido VARCHAR2, p_porcentaje_descuento NUMBER)
    RETURN NUMBER IS
    v_coste_total NUMBER;
  BEGIN
    -- cálculo del coste total del pedido con descuento
    -- Aquí va el código para calcular el coste total del pedido con el número de pedido p_num_pedido
    -- aplicando el descuento p_porcentaje_descuento

    RETURN v_coste_total;
  END CalcularDescuento;
END DescuentoPedidoPackage;
/
```

Activar V

22 - Estructura del paquete.

```
SQL> CREATE OR REPLACE PACKAGE BODY DescuentoPedidoPackage AS
  2  FUNCTION CalcularDescuento(p_num_pedido VARCHAR2)
  3    RETURN NUMBER IS
  4    v_coste_total NUMBER;
  5  BEGIN
  6    SELECT SUM(CANTIDAD * PRECIO) INTO v_coste_total
  7    FROM LINEA_PEDIDO lp
  8    JOIN PRODUCTOS p ON lp.COD_PRODUCTO = p.CODIGO
  9    WHERE lp.NUM_PEDIDO = p_num_pedido;
 10
 11    RETURN v_coste_total;
 12  END CalcularDescuento;
 13
 14  FUNCTION CalcularDescuento(p_num_pedido VARCHAR2, p_porcentaje_descuento NUMBER)
 15    RETURN NUMBER IS
 16    v_coste_total NUMBER;
 17  BEGIN
 18    SELECT SUM(CANTIDAD * PRECIO * (1 - p_porcentaje_descuento/100)) INTO v_coste_total
 19    FROM LINEA_PEDIDO lp
 20    JOIN PRODUCTOS p ON lp.COD_PRODUCTO = p.CODIGO
 21    WHERE lp.NUM_PEDIDO = p_num_pedido;
 22
 23    RETURN v_coste_total;
 24  END CalcularDescuento;
 25 END DescuentoPedidoPackage;
 26 /
```


23 - ¡Paquete creado!

Ejemplo de ambas funciones en acción, invocadas con la nomenclatura del paquete previo a un punto y a la función, pues las estoy llamando desde fuera del paquete.

```
SQL> DECLARE
2  v_coste_total_sin_descuento NUMBER;
3  v_num_pedido VARCHAR2(4) := 'P001';
4  BEGIN
5  v_coste_total_sin_descuento := DescuentoPedidoPackage.CalcularDescuento(v_num_pedido);
6  DBMS_OUTPUT.PUT_LINE('Coste total sin descuento: ' || v_coste_total_sin_descuento);
7  END;
8  /
Coste total sin descuento: 31
SQL>
SQL> -- llamada a la función CalcularDescuento con descuento del 10% aplicado
SQL> DECLARE
2  v_coste_total_con_descuento NUMBER;
3  v_num_pedido VARCHAR2(4) := 'P001';
4  v_porcentaje_descuento NUMBER := 10;
5  BEGIN
6  v_coste_total_con_descuento := DescuentoPedidoPackage.CalcularDescuento(v_num_pedido, v_porcentaje_descuento);
7  DBMS_OUTPUT.PUT_LINE('Coste total con descuento (' || v_porcentaje_descuento || '%): ' || v_coste_total_con_descuento);
8  END;
9  /
Coste total con descuento (10%): 27.9
```

24 - Demostración funciones del paquete.

```
IF INSTR(v_clientes_habituales, 'No se ha encontrado ningún cliente habitual') = 0 THEN
  IF INSTR(v_clientes_habituales, v_num_pedido) > 0 THEN
    v_multiplicador_descuento := 2;
  END IF;
END IF;
```

25 - Complicando...

En ese último trozo de código: Con INSTR se comprueba si v_clientes_habituales no contiene el mensaje "No se ha encontrado ningún cliente habitual". Si el mensaje no está presente, se verifica si la cadena v_clientes_habituales contiene el valor de v_num_pedido. Si v_num_pedido está presente en v_clientes_habituales, se asigna el valor de 2 a la variable v_multiplicador_descuento.

```
CREATE OR REPLACE PROCEDURE CalcularDescuentoProcedimiento AS
v_coste_total_con_descuento NUMBER;
v_num_pedido VARCHAR2(4) := 'P001';
v_porcentaje_descuento NUMBER := 10;
v_multiplicador_descuento NUMBER := 1;
v_clientes_habituales VARCHAR2(200);
BEGIN
v_clientes_habituales := clientes_habituales();

IF INSTR(v_clientes_habituales, 'No se ha encontrado ningún cliente habitual') = 0 THEN
  IF INSTR(v_clientes_habituales, v_num_pedido) > 0 THEN
    v_multiplicador_descuento := 2;
  END IF;
END IF;

v_coste_total_con_descuento := DescuentoPedidoPackage.CalcularDescuento(v_num_pedido, v_porcentaje_descuento *
v_multiplicador_descuento);
DBMS_OUTPUT.PUT_LINE('Coste total con descuento (' || v_porcentaje_descuento || '%): ' || v_coste_total_con_descuento);
END;
```