PROGRAMACION DE BASE DE DATOS ORACLE

Por: Ing. Edwin Calle Terrazas

PL/SQL

PL/SQL es un lenguaje procedimental (Procedural Language extension to SQL).

- Permite desarrollar programas modularizados.
- Programar estructuras de control.
- Manipular errores.
- Programar en la base de datos.
- Soportada por varias herramientas oracle.

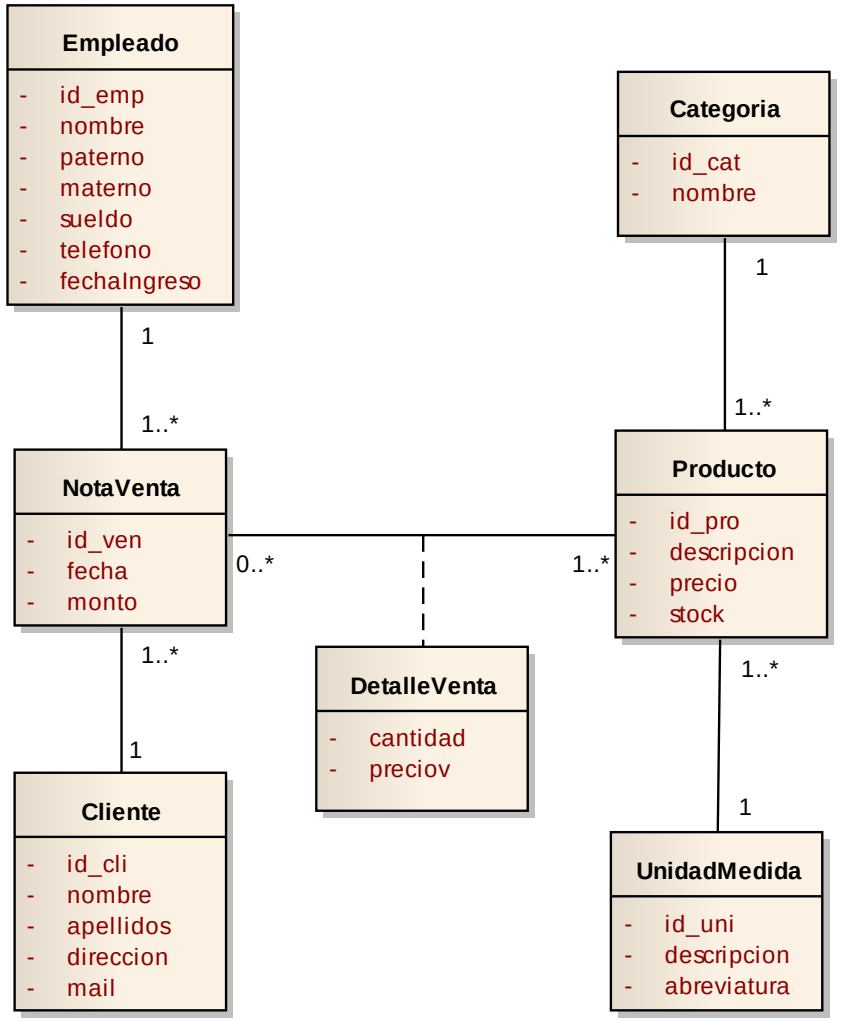
Sin un lenguaje procedimental el servidor debe procesar instrucción por instrucción las solicitudes del cliente, causando un incremento del trafico en la red.

CREACIÓN DE TABLAS PARA LA PRACTICA

```
create table cliente(
id_cli int primary key,
nombre varchar(15) not null,
apellidos varchar(20),
direccion varchar(30),
mail varchar(20)
);

create table empleado(
id_emp int primary key,
nombre varchar(15) not null,
paterno varchar(15) not null,
materno varchar(15),
sueldo number(10,2),
telefono varchar(10),
fechaIngreso date
);

create table categoria(
id_cat int primary key,
nombre varchar(20) not null
);
```



```
create table unidadmedida(
id_uni int primary key,
descripcion varchar(20) not null,
abreviatura varchar(3) not null
);

create table producto(
id_pro int primary key,
descripcion varchar(20) not null,
precio number(8,2) not null,
stock int not null,
id_cat int not null,
id_uni int not null,
foreign key(id_cat) references categoria(id_cat),
foreign key(id_uni) references unidadMedida(id_uni)
);

create table notaventa(
id_ven int primary key,
fecha date not null,
monto number(8,2) not null,
id_cli int not null,
id_emp int not null,
foreign key(id_cli) references cliente(id_cli),
foreign key(id_emp) references empleado(id_emp)
);
```

```
create table detalleventa(
id_ven int not null,
id_pro int not null,
cantidad int not null,
preciov number(8,2) not null,
primary key(id_ven,id_pro),
foreign key (id_ven) references notaventa(id_ven),
foreign key (id_pro) references producto(id_pro)
);
```

CREACIÓN DE SECUENCIAS E INSERCIÓN DE DATOS

```
CREATE SEQUENCE contador_cliente MINVALUE 1 MAXVALUE 9999999999 INCREMENT BY 1 START WITH 1 NOCACHE  NOORDER  NOCYCLE;

insert into cliente values(contador_cliente.nextval,'Carlos', 'Soliz','sucre 501', 'carlo@gmail.com');
insert into cliente values(contador_cliente.nextval,'Maria','Suarez','bolivar 407', 'maria@gmail.com');
insert into cliente values(contador_cliente.nextval, 'Ricardo', 'Tapia', 'arenales 305', 'ricardo@hotmail.com');

insert into empleado values(1,'Carla','Garcia','Lopez',5000,'73145236','10-sep-01');
insert into empleado values(2, 'Jorge', 'Andia', 'Nuñez',3000, '72188555','10-ene-02');
insert into empleado values(3, 'Jose', 'Dias','Lopez',3000,'77820200','20-jun-11');

insert into unidadmedida values(1,'Pieza','pza');
insert into unidadmedida values(2,'Litros','Lt');
insert into unidadmedida values(3,'Kilo','kl');

insert into categoria values(1,'Disco duro');
insert into categoria values(2,'Memoria');
insert into categoria values(3,'Tarjeta Madre');
```

```
insert into producto values(1,'HD Samsung 500 Gb',650,1000,1,1);
insert into producto values(2,'HD Seagate 700 Gb',800,1000,1,1);
insert into producto values(3,'Ram 4 Gb Markvision',150,500,2,1);
insert into producto values(4,'Tarjeta Madre AsRock',400,500,3,1);

CREATE SEQUENCE contador_venta MINVALUE 1 MAXVALUE 99999999
INCREMENT BY 1 START WITH 1 NOCACHE  NOORDER  NOCYCLE;

insert into notaventa values(contador_venta.nextval,'20-sep-14',150,1,2);
insert into detalleventa values(1,1,1,650);
insert into detalleventa values(1,3,1,150);

insert into notaventa values(contador_venta.nextval,'20-sep-14',100,1,2);
insert into detalleventa values(2,2,1,800);
insert into detalleventa values(2,3,2,150);

insert into notaventa values(contador_venta.nextval,'21-sep-14',70,2,1);
insert into detalleventa values(3,2,5,150);

insert into notaventa values(contador_venta.nextval,'21-sep-14',50,1,1);
insert into detalleventa values(4,2,1,650);
```

Crear un usuario y otorgar privilegios de conexión y recursos

```
grant connect, resource to edwin identified by edw;
```

PROGRAMACIÓN CON PL-SQL

SQL es un lenguaje de comandos, no un lenguaje de programación con todas las estructuras de control típicas. Así, SQL sólo contempla instrucciones, más o menos simples, pero no tiene ningún tipo de instrucciones de control de flujo o de otro tipo más propias de los lenguajes de programación 3GL.
Para subsanar esta carencia, Oracle definió un lenguaje de programación que admitía sentencias SQL embebidas, este lenguaje se llama PL/SQL (Programming Language/SQL). La idea básica sobre la que se sustenta el PL/SQL es aplicar las estructuras típicas de un lenguaje de programación (bifurcaciones, bucles, funciones, etc) a las sentencias SQL típicas.

Los comandos más importantes de estos lenguajes son:
LDD, definición de datos:
CREATE TABLE, CREATE INDEX, CREATE VIEW, DROP TABLE, DROP INDEX, DROP VIEW, ALTER TABLE, COMMENT.

LMD, manipulación de datos:
SELECT, INSERT, UPDATE, DELETE.

LCD, control de datos:
GRANT, REVOKE, COMMIT, ROOLBACK.

SELECT

Recupera información de las tablas, seleccionando una o más filas o columnas de una o muchas tablas. La sintaxis completa de la instrucción SELECT es compleja, sus cláusulas principales pueden ser resumidas de la siguiente manera.

```
SELECT columna1, columna2,...
FROM  tabla1, tabla2,…
WHERE condición
GROUP BY columna1,columna2,…
HAVING condición
ORDER BY columna1,columna2,.. [ASC][DESC]
```

FROM	Determina la tabla o tablas de donde se muestras los registros.
WHERE	Establece los criterios de selección de las filas.
GROUP BY	Establece la lista de columnas por las cuales se agruparán los registros
HAVING	Establece los criterios para filtrar los grupos generados por **GROUP BY**
ORDER BY	Permite ordenar los registros por una o más columnas de acuerdo a los requerimientos.

notaventa

ID_VEN	FECHA	MONTO	ID_CLI	ID_EMP
2	20-SEP-13	150	1	1
3	20-SEP-13	150	1	2
4	21-SEP-13	100	1	2
7	21-SEP-13	50	2	1

cliente

ID_CLI	NOMBRE	APELLIDOS	MAIL
10	Carlos	Soliz	carlo@gmail.com
11	Maria	Suarez	maria@gmail.com
14	Ricardo	Tapia	ricardo@hotmail.com

Mostrar el nombre y teléfono de los clientes de nombre carlos

```
select nombre, telefono  from cliente
where nombre= 'carlos';
```

Mostrar el nombre del cliente, fecha y monto de las notas de ventas realizadas por los clientes en fecha 20-sep-14

```
select nombre, fecha, monto
from cliente, notaventa
where cliente.id_cli = notaventa.id_cli and fecha='20-sep-14';
```

OPERADORES DE LA CLAUSULA WHERE

<	Menor que
<=	Menor igual que
>	Mayor que
>=	Mayor igual que
=	Igual a
<>	No igual a
BETWEEN	Entre (Dentro de un intervalo de valores)
LIKE	Como (Coincide con los caracteres)
IN	En (Contenido en una lista de valores)

El operador BETWEEN devuelve todos los valores de los registros que estén entre Los límites especificados (valorInicial y valorFinal)
Mostrar la fecha y monto de las notas de ventas realizadas del 19 de septiembre al 22 de septiembre de 2014.

```
select  fecha, monto from  notaventa
where fecha between '19-sep-14' and '22-sep-14';
```

El operador LIKE hace coincidir los registros de un campo con un formato especificado a través de los siguientes comodines.

_	Indica un solo caracter.
%	Indica un conjunto de caracteres
[ ]	Representa un rango de valores para un caracter

producto

ID_PRO	DESCRIPCION	PRECIO	STOCK	ID_CAT	ID_UNI
1	HD Samsung 500 Gb	650	1000	1	1
2	HD Seagate 700 Gb	800	1000	1	1
3	Ram 4 Gb Markvision	150	500	2	1
4	Tarjeta Madre AsRock	400	500	3	1

Mostrar la descripción y precio de los productos que comienzan con la letra m, la tercer letra también sea m y no importa con que letra termine.

```
select descripcion, precio from producto
where  descripcion like 'm_m%';
```

Mostrar la descripción y precio de los productos donde su nombre empieza con una letra de a hasta d y no me interesa con que letra termina.

```
select descripcion, precio from producto
where  descripcion like '[a-d]%';
```

El operador IN se utiliza para recuperar registros que coinciden con una lista de datos

Mostrar el nombre del cliente, fecha y monto de las notas de ventas realizadas en fecha 20-sep-14, 22-sep-14 y 23-sep-14

```
select nombre, fecha, monto from cliente, notaventa
where cliente.id_cli = notaventa.id_cli
and fecha IN('20-sep-14','22-sep-14' , '23-sep-14');
```

ORDER BY

Indica que ordene los registros recuperados. Puede ordenar por uno o por varios campos.
Para ordenar en orden descendente puede utilizar la palabra clave DESC

```
select nombre, fecha, monto from cliente, notaventa
where cliente.id_cli= notaventa.id_cli and
fecha between '20-sep-14' and '23-sep-14'
order by fecha desc;
```

FUNCIONES DE AGREGACIÓN

Función	Resultado
SUM	Total de todos los valores del campo
COUNT	Cuenta el número de valores de una expresión
COUNT(*)	Cuenta el número de filas seleccionadas
AVG	Media (promedio) de todos los valores de la columna
MAX	Valor máximo (mayor) de una columna
MIN	Valor mínimo (menor) de una columna
STDEV	Calcula la desviación estándar

producto

ID_PRO	DESCRIPCION	PRECIO	STOCK	ID_CAT	ID_UNI
1	HD Samsung 500 Gb	650	1000	1	1
2	HD Seagate 700 Gb	800	1000	1	1
3	Ram 4 Gb Markvision	150	500	2	1
4	Tarjeta Madre AsRock	400	500	3	1

Mostrar el promedio de los precios de los productos

```
select  AVG (precio) from producto;
```

Mostrar el precio máximo y mínimo de los productos

```
select MAX (precio), MIN(precio) from producto;
```

Mostrar la cantidad de productos que tienen precio mayor o igual que 100

```
select COUNT (id_pro) from producto
where precio >= 100;
```

Mostrar la cantidad de productos y la suma de los precios que pertenecen a la categoría Disco duro

```
select COUNT(id_producto) cantidad, SUM(precio) total_precio
from producto, categoria
where categoria.id_cat = producto.id_cat and categoria.nombre='Disco duro';
```

Mostrar el stock (cantidad de productos) que corresponden a la categoria Disco duro

```
select SUM(stock) as total_stock from producto, categoria
where categoria.id_cat=producto.id_cat and nombre='Disco duro';
```

LA CLAUSULA GROUP BY

Se utiliza para realizar agrupaciones de registros por un criterio establecido. Dicha agrupación puede ser por uno o más campos.
La clausula HAVING es equivalente a la cláusula WHERE, pero esta se aplica a consultas agrupadas.

Cuando utilice la cláusula GROUP BY, considere los hechos e instrucciones siguientes:

1. SQL Server produce una columna de valores por cada grupo definido.
2. SQL Server sólo devuelve filas por cada grupo especificado; no devuelve información de detalle.
3. Todas las columnas que se especifican en la cláusula GROUP BY tienen que estar incluidas en la lista de selección.
4. Si incluye la cláusula HAVING después de GROUP BY, SQL Server mostrará los grupos que cumplen dichas condiciones.
5. No utilice la cláusula GROUP BY en columnas que contengan varios valores nulos, porque los valores nulos se procesan como otro grupo.
6. Utilice la palabra clave ALL con la cláusula GROUP BY para presentar todas las filas que tengan valores nulos en las columnas de agregado, independientemente de si las filas cumplen la condición de la cláusula WHERE.

notaventa

ID_VEN	FECHA	MONTO	ID_CLI	ID_EMP
2	20-SEP-13	150	10	2
3	21-SEP-13	100	10	2
4	21-SEP-13	50	11	1

Mostrar la fecha y monto de las notas de ventas realizadas por día

```
select fecha, SUM(monto) total from notaventa
group by fecha;
```

Mostrar el id_cli y la cantidad de veces que pidió una nota de venta

```
select id_cli, COUNT(id_cli) cantidad from notaventa
group by id_cli;
```

Mostrar el id_cli, nombre del cliente y la cantidad de veces que el cliente solicito una nota de venta.

```
select notaventa.id_cli, nombre, COUNT(notaventa.id_cli) cantidad_veces
from notaventa, cliente
where cliente.id_cli = notaventa.id_cli
group by notaventa.id_cli, nombre;
```

LA CLAUSULA HAVING

La cláusula HAVING se utiliza en columnas o expresiones para establecer condiciones en los grupos incluidos en un conjunto de resultados. La cláusula HAVING establece condiciones en la cláusula GROUP BY de una forma muy similar a como interactúa la cláusula WHERE con la instrucción SELECT.

Cuando utilice la cláusula HAVING, considere los hechos e instrucciones siguientes:

1. La cláusula HAVING sólo con la cláusula GROUP BY para restringir los agrupamientos. El uso de la cláusula HAVING sin la cláusula GROUP BY no tiene sentido.
2. En una cláusula HAVING puede haber muchas condiciones. Cuando utilice varias condiciones, tiene que combinarlas con operadores lógicos (AND, OR ó NOT).
3. Puede hacer referencia a cualquiera de las columnas agrupadas que aparezcan en la lista de selección.

Lo mismo que lo anterior, pero mostrando el monto total de una fecha específica

```
select fecha, SUM(monto) total from notaventa
group by fecha
having fecha = '22-sep-14';
```

Mostrar el id_cli y la cantidad de veces que pidieron una nota de venta más de 3 veces.

```
select id_cli, COUNT(id_cli) cantidad from notaventa
group by id_cli
having COUNT(id_cli)>3;
```

CREACIÓN DE VISTAS

Una vista ofrece la posibilidad de almacenar una consulta como un objeto en una base de datos para usarse posteriormente.

Podemos decir que una vista es:
- Un subconjunto de las filas o columnas de una o más tablas.
- Un resumen estadístico de una tabla base.
- Un subconjunto de otra vista o alguna combinación de vistas y tablas base.

Su formato es:

```
CREATE VIEW nombre_vista
AS
-- consulta
```

```
CREATE VIEW monto_fecha
AS
select fecha, SUM(monto)as total from notaventa
group by fecha;
```

Podemos mostrar los datos de la vista

```
select *from monto_fecha
where fecha='22-sep-14';
```

```
CREATE VIEW cliente_con_notaventa
AS
(select id_cli from cliente) intersect (select id_cli from notaventa);
```

Mostrar el nombre de los clientes que realizaron solicitaron notas de ventas

```
select nombre from cliente_con_notaventa, cliente
where cliente_con_notaventa.id_cli=cliente.id_cli;
```

```
CREATE VIEW cliente_sin_notaventa
AS
(select id_cli from cliente) minus (select id_cli from notaventa);
```

Mostrar el nombre y telefono de los clientes que no realizaron ninguna nota de venta

```
select nombre,telefono from cliente_sin_notaventa, cliente
where cliente_sin_notaventa.id_cli=cliente.id_cli;
```

Para eliminar una vista:

```
drop view cliente_sin_notaventa
```

FUNCIONES INTEGRADAS MAS UTILIZADAS

Función	*Cometido*	*Ejemplo*	*Resultado*
ABS(n)	Calcula el valor absoluto de *n*.	select abs(-15) from dual;	15
CEIL(n)	Calcula el valor entero inmediatamente superior o igual a *n*.	select ceil(15.7) from dual;	16
FLOOR(n)	Calcula el valor entero inmediatamante inferior o igual a *n*.	select floor 15.7) from dual;	15
MOD(m,n)	Calcula el resto resultante de dividir *m* entre *n*	select mod(11,4) from dual;	3
POWER(m,n)	Calcula la potencia *n*-esima de *m*.	select power(3,2) from dual;	9
ROUND(m,n)	Calcula el redondeo de *m* a *n* decimales. Si *n*<0 el redondeo se efectúa a por la izquierda del punto decimal.	select round(123.456,1) from dual;	123.5
SQRT(n)	Calcula la raíz cuadrada de *n*	select sqrt(4) from dual;	2
TRUNC(m,n)	Calcula *m* truncado a *n* decimales (*n* puede ser negativo).	select trunc(123.456,1) from dual;	123.4
SIGN(n)	Calcula el signo de *n*, devolviendo -1 si *n*<0, 0 si *n*=0 y 1 si *n*>0	select sign(-12) from dual;	-1

Función	*Cometido*	*Ejemplo*	*Resultado*
CHR(n)	Devuelve el carácter cuyo valor codificado es *n*.	select chr(65) from dual;	A
ASCII(cad)	Devuelve el valor ascii de *cad*.	select ascii('A') from dual;	65
CONCAT(cad1,cad2)	Devuelve *cad1* concatenada con *cad2*. Esta función es esquivalente al operador ||	select concat(concat(nombre,' es '),oficio) from emp;	Cano es Presidente, etc.
LOWER(cad)	Devuelve la cadena *cad* con todas sus letras convertidas a minúsculas.	select lower('MinUsCulAs') from dual;	minusculas
UPPER(cad)	Devuelve la cadena *cad* con todas sus letras convertidas a mayúsculas.	select upper('maYuSCulAs') from dual;	MAYUSCULAS
INITCAP(cad)	Devuelve *cad* con el primer caracter en mayúsculas.	select initcap('isabel') from dual;	Isabel
LPAD(cad1,n,cad2)	Devuelve *cad1* con longitud *n*, y ajustada a la derecha, rellenando por la izquierda con *cad2*	select lpad('P',5,'*') from dual;	****P
RPAD(cad1,n,cad2)	Devuelve *cad1* con longitud *n*, y ajustada a la izquierda, rellenando por la derecha con *cad2*	select rpad('P',5,'*') from dual;	P****
REPLACE(cad,ant,nue)	Devuelve *cad* en la que cada ocurrencia de la cadena *ant* ha sido sustituida por la cadena *nue*.	select replace('digo','i','ie') from dual;	diego
SUBSTR(cad,m,n)	Devuelve la sudcadena de *cad* compuesta por *n* caracteres a partir de la posicion *m*	select substr('ABCDEFG',3,2) from dual;	CD
LENGTH(cad)	Devuelve la longitud de *cad*	select length('cadena') from dual;	6

Función	*Cometido*	*Ejemplo*	*Resultado*
SYSDATE	Devuelve la fecha y hora actuales	select sysdate from dual;	14-MAR-97
ADD_MONTHS(d,n)	Devuelve la fecha *d* incrementada en *n* meses	select add_months(sysdate,4) from dual;	14-JUL-97
LAST_DAY(d)	Devuelve la fecha del último día del mes de *d*	select last_day(sysdate) from dual;	31-MAR-97
MONTHS_BETWEEN(d1, d2)	Devuelve la diferencia en meses entre las fechas *d1* y *d2*	select months_between(sysdate,'01-ENE-97') from dual;	2.43409424
NEXT_DAY(d,cad)	Devuelve la fecha del primer día de la semana *cad* después de la fecha *d*	select next_day(sysdate, 'sunday') from dual;	16-MAR-97

Funciones de Conversión de Tipos

Función	***Cometido***	***Ejemplo***	***Resultado***
TO_NUMBER(cad,fmto)	Convierte la cadena *cad* a un número, opcionalmente de acuerdo con el formato *fmto*	select to_number('12345') from dual;	124345
TO_CHAR(d, fmto)	Convierte la fecha *d* a una cadena de caracteres, opcionalmente de acuerdo con el formato *fmto*	select to_char(sysdate) from dual;	'14-MAR-97'
TO_DATE(cad,fmto)	Convierte la cadena *cad* de tipo varchar2 a fecha, opcionalmente de acuerdo con el formato *fmto*	select to_date('1-JAN-97') from dual;	01-JAN-97

Máscaras de Formato Numéricas

Formato	***Cometido***	***Ejemplo***	***Resultado***
cc ó scc	Valor del siglo	select to_char(sysdate,'cc') from dual;	20
y,yyy ó sy,yyy	Año con coma, con o sin signo.	select to_char(sysdate,'y,yyy') from dual;	1,997
yyyy ó yyy ó yy ó y	Año sin signo con cuatro, tres, dos o un dígitos	select to_char(sysdate,'yyyy') from dual;	1997
q	Trimestre	select to_char(sysdate,'q') from dual;	1
ww ó w	Número de la semana del año o del mes	select to_char(sysdate,'ww') from dual;	11
mm	Número del mes.	select to_char(sysdate,'mm') from dual;	03
ddd ó dd ó d	Número del día del año, del mes o de la semana.	select to_char(sysdate,'ddd') from dual;	073
hh ó hh12 ó hh24	La hora en formato 12h. o 24h.	select to_char(sysdate,'hh') from dual;	12
mi	Los minutos de la hora	select to_char(sysdate,'mi') from dual;	15
ss ó sssss	Los segundos dentro del minuto, o desde las 0 horas.	select to_char(sysdate,'sssss') from dual;	44159

Máscaras de Formato de Caracteres

Formato	***Cometido***	***Ejemplo***	***Resultado***
syear ó year	Año en Inglés	select to_char(sysdate,'syear') from dual;	nineteen ninety-seven
month o mon	Nombre del mes o su abreviatura de tres letras	select to_char(sysdate,'month') from dual;	march
day ó dy	Nombre del día de la semana o su abreviatura de tres letras.	select to_char(sysdate,'day') from dual;	friday
a.m. ó p.m.	El espacio del día	select to_char(sysdate,'a.m.') from dual;	p.m
b.c ó a.d	Indicador del año respecto al del nacimiento de Cristo	select to_char(sysdate,'b.c') from dual;	a.d

Otras Funciones

Función	***Cometido***	***Ejemplo***	***Resultado***
DECODE(var, val1, cod1, val2, cod2, ..., defecto)	Convierte el valor de *var*, de acuerdo con la codificación.	select decode('oficio', 'Presidente', 'P', 'Director', 'D', 'X') from dual;	D X
GREATEST(exp1, exp2, ...)	Devuelve el mayor valor de una lista.	select greatest(23,5,75,2) from dual;	75
LEAST(cad,fmto)	Devuelve el menor valor de una lista.	select least(23,5,80,22) from dual;	5
NVL(val, exp)	Devuelve la expresión exp si *val* es NULL, y *val* si en otro caso.	select nvl(45,0) from dual;	45

Otros ejemplos

```
SELECT CONCAT(nombre ||' ',paterno ||' '|| materno) "nombre completo",
(to_char(sysdate,'yyyy')-to_char(fechaingreso,'yyyy'))  " antiguedad " from empleado;

SELECT sueldo,
ABS(sueldo) "absoluto",
fechaingreso "fecha 1",
ADD_MONTHS(fechaingreso,1) fecha2,
TO_CHAR(fechaingreso,'MM-DD-YYYY') fecha3,
TO_CHAR(fechaingreso,'YYYY') "año",
TO_NUMBER(TO_CHAR(fechaingreso,'MM')) "mes"
FROM empleado;

SELECT
  SQRT(5) "Raiz Cuadrada",
  SUBSTR('ABCDEFG',3,4) "SubCadena",
  TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS')"AHORA",
  TO_CHAR(SYSDATE, 'Month DD, YYYY') "Fecha"
FROM DUAL;

SELECT RTRIM('123Curso Oracle123','123') "RTRIM()", LTRIM('123Curso
Oracle123','123') "LTRIM()", TRIM(' Curso Cracle ') "TRIM()"  FROM DUAL;
```

FUNCIONES DEFINIDAS POR EL USUARIO

Una función es un conjunto de instrucciones en PL/SQL, que pueden ser llamados usando el nombre con que se le haya creado.
Se diferencian de los procedimientos, en que las funciones retornan un valor al ambiente desde donde fueron llamadas.

La sintaxis para crear una función es la siguiente:

```
CREATE [OR REPLACE] FUNCTION nombre (param1 IN datatype), . . .)
RETURN datatype IS
BEGIN

 pl/sql_subprogram

RETURN VALOR;
END;
```

Hacer una función que retorne el mayor de dos números introducidos por parámetro

```
CREATE or REPLACE FUNCTION mayor(a number, b number)
RETURN NUMBER IS
BEGIN
IF (a>b) THEN
   return a;
ELSE
   return b;
END IF;
END;
/
```

Llamada:

```
select mayor(3,7) from dual;
```

Hacer una función que retorne el cubo de un número introducido por parámetro

```
CREATE or REPLACE FUNCTION cubo(a number)
RETURN NUMBER IS
begin
   return a*a*a;
end;
/
```

Hacer una función que calcule el factorial de un número introducido por parámetro

```
CREATE OR REPLACE FUNCTION factorial(N NUMBER)
RETURN NUMBER IS
F NUMBER;
I NUMBER;
BEGIN
F:=1;
I:=1;
WHILE (I<=N) LOOP
   F:=F*I;
   I:=I+1;
END LOOP;
  RETURN F;
END;
/
```

Llamada:

```
select factorial(3) from dual;
```

```
CREATE OR REPLACE FUNCTION factorial2(N NUMBER)
RETURN NUMBER IS
F NUMBER;  I NUMBER;
BEGIN
F:=1;    I:=1;
FOR I IN 1..N LOOP
   F:=F*I;
END LOOP;
  RETURN F;
END;
/

CREATE OR REPLACE FUNCTION factorial3(N NUMBER)
RETURN NUMBER IS
F NUMBER; I NUMBER;
BEGIN
F:=1;   I:=1;
LOOP
   F:=F*I;
   I:=I+1;
   Exit when I>N;
END LOOP;
  RETURN F;
END;
/
```

Hacer una función que retorne el año de antiguedad que tiene un empleado, introduciendo su codigo por parámetro.

```
create or replace function devolver_antiguedad(id_e NUMBER)
RETURN NUMBER IS
fecha date;
BEGIN
select fechaingreso into fecha from empleado where id_emp=id_e;
return to_char(sysdate,'yyyy') - to_char(fecha,'yyyy');
END;
/
```

Llamada:

```
select devolver_antiguedad(2) from dual;
```

Mostrar el total_monto en dinero de un producto dado su codigo por parámetro

```
CREATE OR REPLACE FUNCTION monto_total(id_cl number, id_ve  number)
RETURN NUMBER IS
total NUMBER;
begin
 select sum(preciov * cantidad) into total
 from cliente,notaventa,detalleventa
 where cliente.id_cli=notaventa.id_cli and
 notaventa.id_ven=detalleventa.id_ven
 and cliente.id_cli=id_cl and notaventa.id_ven=id_ve;
 return total;
end;
/
```

Llamada:

```
select monto_total(10,2) from dual;
```

PROCEDIMIENTOS ALMACENADOS

Un procedimiento almacenado es un conjunto de instrucciones en PL/SQL, que pueden ser llamado usando el nombre que se le haya asignado.

La sintaxis para crear un procedimiento es la siguiente:

```
CREATE [OR REPLACE] PROCEDURE name (param1 [IN|OUT] datatype,..)
IS
BEGIN

 pl/sql_subprogram

END;
```

Procedimiento almacenado que cambia el sueldo de un empleado introduciendo el codigo por parámetro.

```
CREATE OR REPLACE PROCEDURE cambiar_sueldo(
id_e IN int,  suel IN number)
IS
BEGIN
  UPDATE empleado set sueldo = suel
    WHERE id_emp = id_e;
  COMMIT WORK;
END;
/
```

Llamada:

```
execute cambiar_sueldo(10,5000);
```

Otra forma de llamar:

```
begin
cambiar_sueldo(10,5000);
end
```

Procedimiento almacenado que inserta los datos de un cliente

```
create or replace procedure insertar_cliente(
nom in varchar,
ape in varchar,
dir in varchar,
mai in varchar)
is
begin
insert into cliente(id_cli,nombre,apellidos,direccion,mail)
values(contador_cliente.nextval, nom,ape,dir,mai);
end;
/
```

Procedimiento almacenado que elimina un cliente introduciendo su codigo por parámetro.

```
create or replace procedure eliminar_cliente(cod in int)
as
begin
delete from cliente where id_cli=cod;
end;
/
```

Procedimiento que modifica los datos del cliente que introduce su codigo por parametro.

```
create or replace procedure
modificar_cliente(
id_c in int,
nom in varchar,
ape in varchar,
dir in varchar,
mai in varchar)
is
begin
update cliente set nombre=nom,
apellidos=ape, direccion=dir, mail=mai
where id_cli=id_c;
end;
/
```

TRIGGERS (DESENCADENADORES)

Un disparador (o *trigger*) es un procedimiento almacenado asociado a una tabla que se ejecuta al realizar una operación "básica" (INSERT, un DELETE o un UPDATE) sobre ésta. La operación básica que despierta al *trigger* es conocida como sentencia disparadora. La ejecución del disparador puede ser antes o después de llevar a cabo la sentencia disparadora. Es posible especificar condiciones adicionales para la ejecución del disparador. Dado que una sentencia disparadora puede afectar una o más filas de una tabla, es necesario especificar si se quiere que el disparador se ejecute para cada una de las filas afectadas o para el bloque en general.

La sintaxis para crear un trigger es la siguiente:

```
CREATE [OR REPLACE] TRIGGER
{BEFORE|AFTER} {DELETE|INSERT|UPDATE [OF col1, col2, . . ., colN]
[OR {DELETE|INSERT|UPDATE [OF col1, col2, . . ., colN]. . .]}
ON table [REFERENCING OLD AS oldname, NEW as newname]
[FOR EACH ROW [WHEN (condition)]]
BEGIN
  pl/sql_block
END;
/
```

- Dependiendo del tipo de sentencia activadora, algunos nombres de correlación pueden no tener sentido. En un *trigger* activado por un INSERT sólo tiene sentido hablar de los valores nuevos. En un *trigger* activado por un UPDATE tiene sentido hablar de los valores nuevos y los viejos tanto para los *before triggers* como para los *after triggers*. En un *trigger* activado por un DELETE, sólo tiene sentido hablar de los valores viejos.
- Los valores nuevos son referenciados utilizando “:new.” antes del nombre de la columna, mientras que los viejos utilizan “:old.” (los “:” son necesarios dentro del bloque de PL/SQL para indicar que es un valor que viene de afuera de la expresión SQL).
- La expresión en una cláusula WHEN de un *row trigger* puede incluir nombres de correlación. Note que los calificadores se usan sin los “:”.
- Si un *before row trigger* cambia el valor de algún “:new.columna”, un *after row trigger* para la misma sentencia activadora verá el cambio realizado por el *before row trigger*.
- Si un *trigger* puede ser activado por más de un tipo de operación (por ejemplo, "INSERT OR DELETE OR UPDATE OF Cliente"), el cuerpo del *trigger* puede utilizar los predicados condicionales INSERTING, DELETING y UPDATING para ejecutar bloques específicos de código, dependiendo del tipo de operación que activó el disparador. Por ejemplo, si se tiene

```
INSERT OR UPDATE ON Tabla
```

dentro del código del *trigger* se pueden incluir las siguientes condiciones:

```
IF INSERTING THEN . . . END IF;
IF UPDATING THEN . . . END IF;
```

la primera condición es cierta sólo si la operación que disparó el *trigger* es un INSERT. La segunda condición es cierta sólo si la operación que disparó el *trigger* es un UPDATE.

```
CREATE or REPLACE TRIGGER disminuir_stock
BEFORE INSERT on detalleventa
FOR EACH ROW
BEGIN
 IF INSERTING then
  UPDATE producto set stock = stock - :new.cantidad
   where  id_pro = :new.id_pro;
 END IF;
END;
/
```

```
CREATE OR REPLACE TRIGGER calcular_monto
BEFORE INSERT ON detalleventa
FOR EACH ROW
when (new.id_ven > 0)
DECLARE
suma number;
BEGIN
SELECT SUM(cantidad * preciov) into suma from detalleventa
WHERE id_ven = :new.id_ven;
 suma:=suma + (:new.cantidad * :new.preciov);
 UPDATE notaventa set monto = suma
 where id_ven = :new.id_ven;
END;
/
```

```
CREATE O REPLACE TRIGGER tbl_notaventa
AFTER  INSERT  OR  DELETE  OR  UPDATE ON notaventa
BEGIN
IF INSERTING  THEN
   dbms_output.put_line('Se ha creado la nota de venta');
END IF;

IF DELETING  THEN
   dbms_output.put_line('Se ha eliminado la nota de venta');
END IF;

IF UPDATING  THEN
   dbms_output.put_line('Se ha modificado la nota de venta');
END IF;
END;
/
```

MANEJO DE CURSORES

Cuando dentro de un intérprete SQL escribimos una consulta SELECT, el intérprete nos muestra las distintas filas del resultados para que podamos verlas. Sin embargo, dentro de un lenguaje de programación tenemos un problema, ya que lo más común no es mostrar el resultado, sino almacenarlo en variables para su posterior tratamiento.

Para realizar una consulta a través de un cursor hay que realizar los siguientes pasos:

1.- Declarar el cursor (dentro de la sección DECLARE)
2.- Abrir el cursor en el servidor
3.- Recuperar cada una de sus filas (bucle)
4.- Cerrar el cursor

1.- DECLARAR EL CURSOR

En este paso se define el nombre que tendrá el cursor y qué consulta SELECT ejecutará. La sintaxis básica es:

DECLARE
CURSOR nombre_cursor IS
SELECT . . .
FROM . . .

2.- Abrir el cursor en el servidor

Un cursor internamente es una sentencia SELECT cuyo resultado se guarda en el servidor en tablas temporales y que se va retornando cada una de las filas según se va pidiendo desde el cliente. La apertura del cursor debe realizarse sólo una vez.

La sintaxis de apertura de un cursor es:

OPEN nombre_cursor;

Una vez que el cursor está abierto, se podrá empezar a pedir los resultados al servidor.

3.- Recuperar cada una de sus filas

Una vez que el cursor está abierto en el servidor se podrá hacer la petición de recuperación de fila. Este paso es equivalente a hacer una consulta SELECT de una sola fila, ya que estamos seguros de que no vamos a recuperar más de una fila.

La sintaxis de recuperación de fila de un cursor es:

FETCH nombre_cursor INTO variable_fila;

Podremos recuperar filas mientras la consulta SELECT tenga filas pendientes de recuperar. Para saber cuándo no hay más filas podemos consultar los siguientes atributos de un cursor:

Nombre de atributo	Retorna	Descripción
Nombre_cursor%FOUND	BOOLEAN	Retorna si la última fila recuperada fue válida
Nombre_cursor%ISOPEN	BOOLEAN	Retorna si el cursor está abierto
Nombre_cursor%NOTFOUND	BOOLEAN	Retorna si la última fila fue inválida
Nombre_cursor%ROWCOUNT	NUMBER	Retorna el número de filas recuperadas

4.- Cerrar el cursor

Una vez que se han recuperado todas las filas del cursor, hay que cerrarlo para que se liberen de la memoria del servidor los objetos temporales creados. Si no cerrásemos el cursor, la tabla temporal quedaría en el servidor almacenada con el nombre dado al cursor y la siguiente vez ejecutásemos ese bloque de código, nos daría la excepción CURSOR_ALREADY_OPEN (cursor ya abierto) cuando intentásemos abrir el cursor.

Para cerrar el cursor se utiliza la siguiente sintaxis:

CLOSE nombre_cursor;

Actualizar el stock de todos los productos a través valor introducido por teclado, utilizando cursores.

```
CREATE OR REPLACE PROCEDURE añadir_stock(cant IN NUMBER)
IS
cantidad number;
CURSOR micursor IS
    SELECT *from producto;
    datofila micursor%rowtype;
BEGIN
  OPEN micursor;
  FETCH micursor INTO datofila;
   WHILE micursor%found LOOP
      IF datofila.stock > 0 THEN
       cantidad:=datofila.stock + cant;
       UPDATE producto set stock=cantidad
       where id_pro=datofila.id_pro;
   END IF;
   FETCH micursor INTO datofila;
  END LOOP;
   IF micursor%ISOPEN THEN
    CLOSE micursor;
  END IF;
END;
/
```

Función que devuelve el monto total vendido de una fecha introducida x parámetro

```
CREATE OR REPLACE FUNCTION total_fecha(vfecha in date)
RETURN number IS
total number;
 CURSOR micursor IS
    SELECT (preciov*cantidad) subtotal FROM notaventa,cliente,detalleventa
    WHERE notaventa.id_cli=cliente.id_cli and
    detalleventa.id_ven=notaventa.id_ven and notaventa.fecha=vfecha;
 datofila micursor%rowtype;
BEGIN
  total:=0;
  OPEN micursor;
  FETCH micursor INTO datofila;
  WHILE micursor%FOUND LOOP
     IF datofila.subtotal > 0 THEN
        total:= total + datofila.subtotal;
     END IF;
    FETCH micursor INTO datofila;
  END LOOP;
  IF micursor%isopen THEN
    CLOSE micursor;
  END IF;
 return total;
END;
/
```

Aumenta un porcentaje si el empleado tiene una antigüedad > 5años

```
CREATE OR REPLACE PROCEDURE aumentar_bono(valor in number)
IS
cantidad number;
 CURSOR micursor IS
  select id_emp, sueldo, to_char(sysdate,'yyyy') - to_char(fechaingreso,'yyyy') antiguedad
  from empleado;
 datofila micursor%rowtype;
BEGIN
  cantidad:=0;
  OPEN micursor;
  FETCH micursor INTO datofila;
  WHILE micursor%FOUND LOOP
     IF datofila.antiguedad >= 5  THEN
      cantidad:=datofila.sueldo + (datofila.sueldo * valor)/100;
      UPDATE empleado set sueldo=cantidad where id_emp=datofila.id_emp;
     END IF;
   FETCH micursor INTO datofila;
  END LOOP;
  IF micursor%ISOPEN THEN
    CLOSE micursor;
  END IF;
END;
/
```

Procedimiento que vende un producto a través de sus parámetros

```
CREATE OR REPLACE PROCEDURE vender( id_v IN NUMBER, id_c IN NUMBER,
 id_e IN NUMBER, id_p IN NUMBER, cant IN NUMBER)  IS
cantidad NUMBER;
CURSOR micursor IS
    SELECT *FROM producto WHERE id_pro=id_p;
 datofila micursor%ROWTYPE;
BEGIN
  OPEN micursor;
  FETCH micursor INTO datofila;
  WHILE micursor%FOUND LOOP
     IF datofila.stock > 0 THEN
       cantidad:= datofila.stock - cant;
       UPDATE producto set stock=cantidad where id_pro=id_p;
     END IF;
    FETCH micursor INTO datofila;
  END LOOP;
  INSERT INTO notaventa(fecha,id_cli,id_emp) values(sysdate,id_c,id_e);
  INSERT INTO detalleventa VALUES(id_v,id_p,cant,datofila.precio);
   IF micursor%ISOPEN THEN
    CLOSE micursor;
  END IF;
END;
/
```