Oracle básico (I): Creación y manejo de tablas

En el artículo anterior (ver ***Algoritmo*** nº 7, págs. 5-10) dimos un vistazo general a *Oracle* como lenguaje de programación. Con esta entrega iniciamos una serie de artículos sobre aquellos aspectos que consideramos básicos para iniciarse en la programación en Oracle, entre los cuales destacamos los siguientes:

- Creación y manejo de tablas
- Consultas con SQLPlus
- Pantallas de ingreso con SQLForms
- Programación en PL/SQL
- Informes con SQLReport
- Diseño de menú con SQLMenu

Ajustándonos a esta lógica, el tema central de este artículo y punto de partida en cualquier curso de Oracle será el diseño y creación de tablas.

Creación de tablas

Como expusimos en nuestro artículo anterior, en Oracle cada estructura de información se denomina TABLA las cuales, junto a los índices y al diccionario de datos del sistema, componen la base de datos. Por lo tanto, la creación de las tablas en el proceso de programación en Oracle juegan un papel muy importante. En el momento de crear las tablas se definen características a dos niveles: Tabla y Columna, como se muestra a continuación:

A nivel de Tabla

Nombre: Nombre de la tabla puede ser de 1 a 30 caracteres.

Propietario: La tabla tiene como propietario al usuario que las crea En nuestro caso somos el usuario *EIDOS*. Otro usuario que desee usar nuestras tablas debe tener autorización para ello y hacer referencia a la tabla como *eidos.clientes* (*propietario.tabla*)

Cantidad de Columnas: Una tabla puede tener un máximo de 254 columnas.

A nivel de Columna

Nombre: Puede tener de 1 a 30 caracteres.

Tipo de dato y su ancho

CHAR	Máximo de 255. Por defecto 1.
NUMBER	Máximo de 105 dígitos. Por defecto 44.
INTEGER	Numérico sin decimal. Por defecto 38.
DATE	Hasta el 31 de diciembre de 4712.
LONG	Tipo caracter con tamaño variable hasta 65535 bytes. Permite una sola columna LONG por tabla. No se puede usar en subconsultas, funciones o índices.
RAW	Dato en binario puro (imágenes y sonido) con un ancho máximo de 255.
LONGRAW	Igual que *LONG*, pero para almacenar datos en binario puro.

Restricciones: Su función es definir reglas de validación de la columna.

Para facilitar la continuidad del análisis, usaremos como ejemplo las tablas definidas en el artículo anterior: *Clientes* y *VENTAS*.

La definición de *restricciones* al crear las tablas permite establecer reglas de validación de datos, así como los controles necesarios para mantener la integridad referencial entre tablas a través de las columnas claves. Las restricciones que se pueden definir son:

Valor obligatorio: En Oracle existe el concepto de valor nulo (*NULL*), como un valor indefinido o ausencia de valor y que es diferente al numero 0 o al carácter espacio. Por lo tanto, para que una columna siempre tenga valor (sea obligatoria) se define como *NOT NULL*.

Rango de valores: Sirven para chequear que el valor sea mayor a un valor determinado o para que se encuentre entre dos valores.

Clave Primaria: Columnas que identifican de forma única al registro, es un valor único y no nulo (*NOT NULL*). Por ejemplo: el código del cliente es una *clave primaria* que identifica de forma única e irrepetible a cada cliente.

Clave Externa: Columna de la tabla que hace referencia a un valor que tiene que estar registrado en otra tabla. Por ejemplo: la columna código de la tabla VENTAS es una clave externa que hace referencia a un valor de la columna código (clave primaria) de la tabla *Clientes*.

En la versión 6 de Oracle (que dado lo reciente de la versión 7 aún se usa ampliamente) la única restricción que estaba activa era la de valor obligatorio (*NOT NULL*), siendo las otras restricciones sólo declarativas, o sea, que quedaban registradas en la definición de la tabla, pero no se podían activar. En la versión 6, para garantizar la unicidad de la clave primaria, era necesario crear índices con claves únicas, aspecto éste que retomaremos más adelante. En la versión 7 de Oracle estas restricciones están implementadas, garantizándose la verificación y correción de datos en cualquier momento sin tener que programar estos controles.

Destacadas estas cuestiones veamos, entonces cómo se procede para crear las tablas *Clientes* y *Ventas*.

Tabla *Clientes*

Objetivo: Ficha con datos para identificar al cliente. Consta del código del cliente (número secuencial), fecha de alta al sistema, nombre, teléfono, dirección y alguna anotación.

Requisitos: Se debe identificar a cada cliente con un código único (clave primaria), registrando su nombre, teléfono y fecha de registro (estos datos son obligatorios). La dirección y anotaciones son campos opcionales.

Creación de la tabla: Ver el fuente 1

```
Fuente 1
CREATE TABLE clientes
(
      codigo     integer   NOT NULL
                                 PRIMARY KEY,
      fecha      date      NOT NULL,
      nombre     char(30)    NOT NULL,
      telefono   char(20)    NOT NULL,
      direccion  char(100),
      anotacion  LONG
```

```
      );
```

Tabla *Ventas*

Objetivo: Registrar las ventas con al siguiente información: Código del cliente, fecha de la venta, artículo y valor de la venta.

Requisitos: El número del cliente es una clave externa que hace referencia a la columna codigo en la tabla Clientes. En este caso, todos los datos son obligatorios. Se controla que la columna valor sea mayor a cero.

Creación de la tabla: ver fuente 2

```
Fuente 2
CREATE TABLE ventas
(
      codigo   INTEGER    NOT NULL
      REFERENCES clientes(codigo),
      fecha    DATE         NOT NULL,
      articulo CHAR(20)     NOT NULL,
      valor    NUMBER(10,2) NOT NULL
                              CHECK (valor>0)
);
```

Las restricciones de Claves Primaria y Clave Externa se definieron a nivel de columna, pero se pueden definir a nivel de tabla, al final de la misma, como se muestra en el fuente 3:

```
Fuente 3
CREATE TABLE clientes
(codigo     INTEGER      NOT NULL,
nombre      CHAR(30)     NOT NULL,
direccion   CHAR(100),
anotacion   LONG,
PRIMARY KEY (codigo));

CREATE TABLE ventas
(codigo     INTEGER      NOT NULL,
 fecha      DATE         NOT NULL,
 articulo   CHAR(10),
 valor      NUMBER(6,2)  NOT NULL
            CHECK (valor>0),
FOREIGN KEY (codigo)  REFERENCES clientes(codigo));
```

La definición de la clave a nivel de tabla es necesaria cuando la misma está formada por más de una columna.

Unicidad de la clave con índices

Para garantizar la unicidad de los valores de la *clave primaria* de la tabla *Clientes* (en la versión 6 donde esta restricción sólo es declarativa y no está activa), se debe crear un índice que garantice la unicidad de la clave principal. Un requisito importante para la unicidad de la clave principal es que las columnas de la clave se definen como *NOT NULL*.

A continuación mostraremos cómo crear el índice *cliente_codigo* para garantizar la unicidad de la clave primaria:

```
CREATE UNIQUE INDEX cliente_codigo
       ON clientes(código);
```

Secuencias: codificación numérica

La codificación numérica del cliente se puede realizar con una secuencia que automáticamente genera los números enteros en orden ascendente, no siendo necesario recordar cuál fue el último número asignado ; esto evita la duplicidad de códigos.

La secuencia es un objeto que genera valores enteros únicos y se emplean para crear claves primarias numéricas, con el uso del siguiente mandato:

```
CREATE SEQUENCE codigo_cliente
             INCREMENT BY 1
             START WITH 1;
```

Para registrar un nuevo código con la secuencia definida anteriormente se usa la pseudo-columna *codigo_cliente.NEXTVAL,* la cual nos dará el siguiente valor que le corresponde a la secuencia, la forma en que esto se realiza se explicará más adelante, cuando analicemos el ingreso de datos.

Para conocer el valor actual de la secuencia, o sea, el último código asignado, se usa la pseudo-columna *codigo_cliente.CURRVAL*, desde la tabla *DUAL* del sistema, cuyo fin es poder consultar pseudo_columnas (como se muestra a continuación):

```
SELECT user,sysdate,codigo_cliente.currval
FROM DUAL;
```

donde:

1.- *user* es el nombre del usuario
2.- *sysdate* es la fecha del sistema
3.- *codigo_cliente.currval* es el último valor asignado a la secuencia.

Ingreso de datos

Una vez creadas las tablas, índices y secuencias, estamos en condiciones de ingresar datos en la tabla.

El ingreso, modificación y eliminación de registros se realiza fundamentalmente con el diseño de pantallas (*formularios*) desde el módulo *SQLFORMS* (que será tema de análisis específico en otro artículo). No obstante, en este artículo veremos el uso de los mandatos *INSERT UPDATE* y *DELETE*.

Para ingresar un nuevo registro debemos ensayar lo que se muestra en la tabla 1:

Nombre de la Tabla (acciones)	Columnas
INSERT INTO CLIENTES VALUES(codigo_cliente.NEXTVAL, 'PINTURERIAS PROPIOS', '45 67 89' TO_DATE('10/04/95','DD/MM/YY'), 'Uruguay 1234', 'LIBRERIA');	 codigo= secuencia nombre teléfono fecha dirección anotación

Tabla1: Inserción de nuevos registros

Como se podrá observar, en este ejemplo no se especificó la lista de columnas a insertar, lo que indica que se van a ingresar datos para todas las columnas. Por lo tanto, los valores para cada columna se tienen que ingresar en el orden en que están definidos en la tabla. Además, es de destacar que la palabra reservada *VALUES* indica la lista de valores a ingresar; que los datos tipo carácter van entre comillas; que la fecha se registra como una cadena de caracteres usando la función *TO_DATE* (encargada de transformar la cadena de caracteres '10/04/95' en fecha, a partir de un formato de fecha especificado -'DD/MM/YY'-).

También observamos que a la columna *codigo* se le asignó el siguiente valor de la secuencia *codigo_cliente* (*codigo_cliente.NEXTVAL*).

En caso de que sólo se asignaran valores a algunas columnas se debe dar la lista de columnas como se muestra en el fuente 4 correspondiente a la lista de columnas.

```
Fuente 4
INSERT INTO CLIENTES (codigo,nombre,teléfono,fecha)
    VALUES
    (codigo_cliente.NEXTVAL,'CASA AUGE DEPORTES',
    '598768',TO_DATE('15/04/95','DD/MM/YY'));
```

Las columnas a las que se les ingresa información se listan después del nombre de la tabla, en el orden deseado. Las columnas no listadas tendrán valor *NULL*, por ello todas las columnas definidas como obligatorias (*NOT NULL*) deben estar en la lista.

Los siguientes ejemplos muestran posibles errores y sus correspondientes mensajes en el registro de datos:

1.- Falta dato del teléfono que es obligatorio:

```
    INSERT INTO CLIENTES(codigo,nombre)
        VALUES (codigo_cliente.NEXTVAL,
                  'EMPRESA D'
                );
```

Mensaje de error:

ORA-01400: mandatory (NOT NULL) column is missing or NULL during insert.

2.- Intento de registrar cliente con código ya existente:

```
INSERT INTO CLIENTES(codigo,nombre,teléfono)
    VALUES  (1,'EMPRESA TTT','341234');
```

Mensaje de error:
ORA-00001: duplicate key in index

Obsérvese en el caso 2 que la secuencia *codigo_cliente* **no** fue usada al ingresar el valor del código, y sí en el caso 1, provocando error de duplicidad de código. Esto ocurre porque la creación de la secuencia no garantiza la unicidad del código, ya que podemos registrar un código de cliente sin su uso. Sin embargo, la *unicidad* esta garantizada por la definición del índice único visto anteriormente. Si siempre se usa la secuencia la unicidad por supuesto que está garantizada, pero la simple definición de la secuencia no es garantía de su uso.

Listados de registros

A continuación veremos cómo obtener listados para revisar la información registrada, para lo cual seleccionaremos (*select*) registros desde (*from*) una tabla. En realidad el mandato *SELECT* será tema de análisis más detallado en la próxima entrega, por lo que ahora sólo lo trataremos con el objetivo de visualizar los datos ingresados.

Para obtener un listado de todas las columnas y todos los registros de la tabla *Clientes* debemos seguir este procedimiento:

Nombre de la Tabla

```
SELECT * FROM clientes;
```

Donde * = Todas las columnas

El resultado es el que se muestra en la tabla 2:

Código	Fecha	Nombre	Teléfono	Dirección	Anotación
1	10-Apr-95	Pinturerías propios	45 67 89	Uruguay 1234	Ferretería
2	15-Apr-95	Casa Auge deportes	598768		
3	20-Apr-95	Feria del libro	(0567)845677		
4	30-Apr-95	Club de tenis	905877		

Tabla 2: Datos de la tabla *Clientes*

Para listar sólo algunas columnas de la tabla clientes el procedimiento a seguir es:

```
SELECT codigo,nombre FROM clientes;
```

El resultado es el que se muestra en la tabla 3:

Código	Nombre
1	Pinturerías propios
2	Casa Auge deportes
3	Feria del libro
4	Club de tenis

Tabla 3: Datos de la tabla *Clientes* para *Codigo* y *Nombre*

Modificación de registros.

Para modificar valores de la tabla usaremos el mandato *UPDATE*, con el objetivo de modificar el teléfono y la dirección del cliente *Feria del libro*. Para ello, basta con definir:

```
UPDATE clientes
    SET telefono='234567',
    direccion='Andes 945'
WHERE  nombre='Feria del libro';
```

donde:

Clientes es el nombre de la tabla
SET es para indicar el inicio de la lista de columnas y sus nuevos valores.
WHERE garantiza la selección de la fila del cliente.

Es importante destacar que si no se usa la cláusula *WHERE*, se modificará el valor de la columna en todas las filas de la tabla.

Eliminación de registros

La eliminación de registros se realiza con el mandato *DELETE.* El siguiente ejemplo eliminará los clientes con el código cero:

```
DELETE FROM clientes
    WHERE codigo=0;
```

En este caso, si se omite la cláusula *WHERE* serán eliminados todos los registros de la tabla.

SQLPlus

Todas las tareas anteriormente estudiadas se realizan con el módulo *SQLPlus* de Oracle, que trabaja en forma interactiva. A continuación enunciaremos los pasos necesarios para usar SQLPLUS y poder crear tablas, índices o secuencias, así como insertar datos y obtener listados:

1.- Llamar al programa

SQLPLUS

2.- Identificación del usuario

Enter user-name: EIDOS
Enter password:
Si la identificación es correcta se obtiene mensaje de:
Connected to: ORACLE

Si, por el contrario, la identificación es incorrecta se recibe el siguiente mensaje:
ERROR: ORA-01017:
invalid username/password;
logon denied

3.- Indicador en pantalla de que *SQLPlus* está a la espera de la orden:

SQL>_

4.- Escribir los mandatos de creación de tablas, índices y secuencia en un archivo (*TABLAS.SQL*) con el uso del editor.

SQL>edit tablas

5.- Ejecutar los mandatos escritos en el archivo *TABLAS.SQL*

SQL>@tablas

6.- Para salir de *SQLPlus*

SQL>exit

Otras tareas

A continuación examinaremos una serie de mandatos, a nivel de definición de las tablas, gracias a los cuales se puede:

1.- Listar estructuras de las tablas (*DESCRIBE*)
2.- Modificar la estructura de las tablas (*ALTER TABLE*)
3.- Renombrar las tablas (*RENAME*)
4.- Eliminar una tabla (*DROP TABLE*)
5.- Eliminar un indice (*DROP INDEX*)
6.- Consultar las tablas del diccionario
7.- Listado de tablas, Índices y secuencias propiedad del usuario.

Detengámonos en los detalles más significativos de cada una de dichas tareas:

1.- Listar estructura de las tablas (*DESCRIBE*)

Para obtener la estructura (descripción de una tabla) el mandato que se debe emplear es:

```
SQL>DESCRIBE clientes;
```

con lo que el resultado será el que se ofrece en la tabla 4.

Name	Null?	Type
NUMERO	NOT NULL	NUMBER(38)
FECHA	NOT NULL	DATE
NOMBRE	NOT NULL	CHAR(30)

TELEFONO	NOT NULL	CHAR(20)
DIRECCION		CHAR(100)
ANOTACION		LONG

Tabla 4: Resultado del uso del mandato *DESCRIBE*

2.- Modificar la estructura de las tablas (*ALTER TABLE*)

La modificación de la estructura de las tablas con el uso de *ALTER* permite:

- Añadir nuevas columnas.
- Añadir restricciones a una columna, en este caso la tabla no debe contener datos.
- Modificar el ancho de la columna.
- Modificar el tipo de datos de la columna sólo si la columna no contiene datos o está vacía.
- Modificar al tipo LONG sólo una columna sin restricciones.

El siguiente ejemplo muestra cómo añadir, en la tabla *Ventas*, las columnas *Factura* (para registrar el número de factura) y *Cobro* (tipo carácter con 2 posibles valores, N=NO cobrada, NULL=cobrada) y modificar la columna valor para ampliar su ancho.

```
ALTER TABLE ventas
ADD (
    factura   integer,
    cobro     char
)
MODIFY (
    valor number(10,2)
);
```

3.- Renombrar las tablas (*RENAME*)

Para cambiar el nombre de la tabla *Clientes* a E*MPRESAS* se usa el siguiente mandato:

```
SQL>RENAME clientes TO empresas;
```

4.- Eliminar una tabla (*DROP TABLE*)

Le eliminación de la tabla es como sigue:

```
SQL>DROP TABLE clientes;
```

En este caso se eliminan, también, todos los índices de la tabla.

5.- Eliminar un índice (*DROP INDEX*)

```
SQL>DROP INDEX cliente_codigo;
```

6.- Consultar las tablas del diccionario.

Toda la información de las tablas está registrada en el diccionario del sistema (*Data Dictionary*), que son tablas especiales que se crean en la instalación de ORACLE (que son administradas por el sistema).

Para consultar la lista de tablas que componen el diccionario se escribe:

```
SQL>HELP DATA DICT
```

Gracias a lo cual se muestra una lista con la información de la tabla 5:

Nombre de la tabla	Descripción
ACCESSIBLE_COLUMNS	columns of all tables, views, and clusters
ACCESSIBLE_TABLES	tables and views accessible to the user
AUDIT_ACTIONS	maps action type numbers to action type names
ALL_INDEXES	descriptions of indexes on accessible
ALL_SEQUENCES	descriptions of the user's own sequences
ALL_TABLES	description of tables accessible to the user
.....	
USER_TABLES	descriptions of the user's own tables
USER_TAB_COLUMNS	columns of the user's tables, views, and clusters
USER_TAB_GRANTS	grants on objects where the user is the owner, grantor, or grantee

Tabla 5: Consulta de las tablas que componen el diccionario

También podemos ver la estructura de una tabla del diccionario como se muestra a continuación:

```
SQL>DESCRIBE ALL_TABLES;
SQL>DESCRIBE all_indexes;
SQL>DESCRIBE all_sequences;
```

7.- Listar las tablas, índices y secuencias definidas por un usuario

Para las tablas:

```
SQL>SELECT TABLE_NAME "TABLA"
    FROM ALL_TABLES
    WHERE OWNER='EIDOS';
```

Resultado: *Clientes* y *Ventas*

Para los índices:

```
SQL>SELECT table_name,index_name
    FROM all_indexes
    WHERE owner='EIDOS';
```

Resultado: *Clientes* (con índice *Cliente_Nombre* y *Cliente_Numero*) y *Ventas* (con índice *Venta_Numero*)

Para las secuencias:

```
SQL>SELECT sequence_name
    FROM all_sequences
```

```
WHERE sequence_owner='EIDOS';
```

Resultado: el nombre de la secuencia usada (*SEQUENCE_NAME*) *Codigo_Cliente*

Como hemos visto, la creación de las tablas constituye el fundamento del diseño de cualquier sistema a desarrollar en Oracle. Una vez definida las tablas el paso lógico siguiente es conocer las técnicas para realizar un adecuado uso de la información contenida en el sistema. Por ello, el próximo artículo lo dedicaremos al lenguaje de Consulta *SQL*.

Oracle básico (II): Creación y manejo de tablas

Con el artículo anterior iniciamos una entrega de Oracle Básico comenzando con el tema de creación y manejo de tablas. Ahora pasaremos a estudiar la consulta y selección de registros con el lenguaje estándar para bases de datos relacionales *SQL* (*Structured Query Languague = Lenguaje de Consulta estructurado*).

La ventaja principal del SQL, desde mi punto de vista, es su capacidad de combinar sencillez y facilidad con potencia y eficiencia, conteniendo un conjunto de herramientas que optimizan las consultas.

Vale la pena destacar que, aunque los conceptos a estudiar son específico de *ORACLE*, también son útiles para cualquier programador que esté trabajando con algún software que contenga SQL.

Sentencia *SELECT*

Como ya sabemos, la herramienta fundamental de SQL es la sentencia *SELECT*, que permite seleccionar registros desde las tablas de la Base de Datos, devolviendo aquellos que cumplan las condiciones establecidas y pudiendo presentar el resultado en el orden deseado.

Primeramente estudiaremos la forma básica de la sentencia SELECT, que esta formado por:

```
SELECT   Lista...
FROM     Tabla, Tabla...
WHERE    Condiciones...
      ORDER BY Expresión,Expresión,...
      ;      Fin de la sentencia.
```

Donde:

La orden SELECT puede contener:

- Columnas: nombre, telefono
- Expresiones y funciones: *SYSDATE-fecha,UPPER(direccion)*
- Pseudo-Columnas del Sistema: *SYSDATE*,*USER*.
- Asterisco: Todas las columnas.

La orden *FROM* identifica la lista de tablas a consultar. Si alguna de las tablas a consultar no es propiedad del usuario, debe especificarse el nombre del propietario antes que el nombre de la tabla en la forma *nombre_propietario.nombre_tabla*.

La orden *WHERE* decide los registros a seleccionar según las condiciones establecidas, limitando el número de registros que se muestran.

La orden *ORDER BY* indica el orden en que aparece el resultado de la consulta.

Ilustremos lo explicado hasta el momento con el ejemplo del fuente 1, donde consultaremos las ventas realizadas en los últimos 10 días, mostrando el nombre del cliente, artículo vendido y su valor.

```
Fuente 1

SELECT nombre,articulo,valor   Lista nombre del cliente,
                               nombre del artículo y
                               el valor de la venta.
    FROM                       clientes,ventas
                               Tablas con la información de clientes y
                               ventas.
    WHERE clientes.codigo=ventas.codigo
                               Establece la relación, según código de
                               cliente,
                               entre las tablas clientes y ventas.
          and sysdate-ventas.fecha>=10
                               Consulta las ventas de los últimos 10 días.
    ORDER BY nombre            Ordenar el listado por nombre del cliente.
;                              Fin de la sentencia.
```

El resultado de esta sentencia SELECT sería el de la tabla 1:

NOMBRE	ARTICULO	VALOR
CASA AUGE DEPORTES	PAPEL	330.0
CASA AUGE DEPORTES	DISKETTE	33.0
CLUB DE TENNIS	PAPEL	500.5
...		
CLUB DE TENNIS	PAPEL	100.5
FERIA DEL LIBRO	PAPEL	310.0
PINTURERIAS PROPIOS	PAPEL	220.5
PINTURERIAS PROPIOS	DISKETTE	20.5

Tabla 1: Resultados de la sentencia SELECT del fuente 1

Obsérvese que las columnas que tienen el mismo nombre en ambas tablas se diferencian escribiendo el nombre de la tabla antes que el nombre de la columna, como en el caso de *ventas.fecha*, *ventas.codigo* y *clientes.codigo*.

Operadores lógicos

Para construir la condición de la consulta necesitamos conocer los operadores lógicos, por eso a continuación damos una lista de los operadores más usados, agrupados en cuatro grupos:

1. Valor único:Comprueban un valor simple.
2. Lista de valores:Comprueban más de un valor.
3. Combinaciones lógicas:Combinan expresiones lógicas.
4. Negación:Invierte el resultado de la expresión con operadores de valor único o de lista de valores.

Valor único

> < >= <= =

Operadores clásicos de comparación:
mayor, menor, mayor e igual, menor e igual, igual a.

!= <> ^=

Operador "Distinto de" en sus tres formas.

```
IS NULL
```

Comprueba la ausencia de datos (valor nulo).
No se puede usar la comparación = NULL.

```
LIKE
```

Selecciona registros según el reconocimiento de un patrón de consulta.

Lista de valores

```
BETWEEN valor AND valor
```

Comprueba que el valor se encuentre en el rango de valores.

```
IN (valor,...,valor)
```

Verifica si el valor pertenece a la lista de valores.

Combinaciones lógicas.

```
AND
```

Retorna Verdadero si todas las condiciones son verdaderas.

```
OR
```

Retorna Verdadero si alguna de las condiciones es verdadera.

Negación

```
NOT
```

Invierte el resultado de una expresión lógica, por ejemplo.

```
IS NOT NULL
```

```
NOT BETWEEN valor AND valor

NOT IN  (valor,...,valor)

NOT LIKE
```

A continuación mostramos algunas consultas con el uso de diferentes operadores lógicos:

- Clientes a los que no se les ha registrado su dirección.

```
SELECT nombre,telefono
FROM    clientes
WHERE   direccion IS NULL;
```

- Clientes dados de alta en los últimos 10 días.

```
SELECT nombre,direccion,telefono
FROM   clientes
WHERE  fecha BETWEEN sysdate-10
       AND sysdate;
```

- Datos de los Clientes que pertenecen a una lista de clientes.

```
SELECT nombre,direccion,telefono
FROM   clientes
WHERE  nombre
       IN ('PINTURAS','CASA DE DEPORTES')
;
```

- Clientes dados de altas en lo que va del mes y cuyo nombre comience con la letra P u otra letra mayor o su teléfono contenga el código (0567). Ver fuente 2

```
Fuente 2

SELECT nombre,direccion,telefono,fecha
FROM   clientes
WHERE
fecha BETWEEN
       to_date('01/'||to_char(sysdate,'MM/YY'),'DD/MM/YY')
       AND
       sysdate
       AND (nombre  >= 'P' OR  telefono LIKE '%(0567)%');
```

El manejo de fecha es una de las capacidades de mayor variedad e interés en ORACLE por las posibilidades que presenta en el almacenamiento, cálculo y presentación de fechas. Por eso, en el último ejemplo damos un vistazo a algunas funciones útiles en el uso de fechas como son:

```
to_char(sysdate,'MM/YY')
```

Devuelve una cadena de caracteres de la forma mes/año de la fecha actual.

```
'01/'||to_char(sysdate,'MM/YY')
```

Forma la cadena de caracteres *01/mes/año* que representa el primer día del mes. El operador || se usa para unir o concatenar cadenas de caracteres.

```
. . .
to_date('01/'||to_char(sysdate,
                       'MM/YY'
                      ),
        'DD/MM/YY')
. . .
```

Convierte la cadena de caracteres *01/mes/año* al tipo fecha.

Patrón de consulta

Una de las herramientas lógicas más poderosas de SQL es el reconocimiento de un patrón de consulta, instrumento éste que permite la búsqueda por nombre, dirección u otro dato parcialmente recordado. Los patrones de consulta juegan un papel importante en el momento de realizar consultas, ya que es común que necesitemos encontrar un texto y no recordemos exactamente cómo fue ingresado. Con el uso del operador *LIKE* podemos comparar patrones y ubicar un texto, independientemente de la posición en que se encuentre.

Para la definición del patrón de consulta existen dos tipos de caracteres especiales:

% (signo de porcentaje) llamado *comodín*, representa cualquier cantidad de espacios o caracteres en esa posición. Significa que se admite cualquier cosa en su lugar: un caracter, cien caracteres o ningún caracter.

_ (signo de subrayado) llamado *marcador de posición*, representa exactamente una posición e indica que puede existir cualquier caracter en esa posición.

En los fuentes 3, 4 y 5 detallamos tres ejemplos de consulta con el Operador *LIKE*:

Listar los clientes cuya dirección contengan la palabra *URUGUAY* independientemente de su ubicación:

```
Fuente 3

SELECT nombre,direccion,telefono
FROM   clientes
WHERE  direccion LIKE '%URUGUAY%';

Listar los clientes  cuyos teléfonos tienen comienzan con el código de área
el 0722

SELECT nombre,direccion,telefono
FROM   clientes
WHERE  telefono LIKE '(0722)%';
```

Listar los clientes cuyo nombre terminan con la palabra *LIBRO*:

```
Fuente 4

SELECT nombre,direccion,telefono
FROM   clientes
WHERE  nombre LIKE '%LIBRO';
```

Listar los clientes que tengan la palabra *LIBRO* a partir de la 5ª posición en el nombre.

```
Fuente 5

SELECT nombre,direccion,telefono
FROM   clientes
WHERE  nombre LIKE '____LIBRO%';
```

Agrupamiento de datos

SQL proporciona una forma eficiente para manejar la información con el agrupamiento de datos a través de la formación de grupos y las funciones correspondientes, dando la posibilidad de procesar no solo registros individuales como hemos hecho hasta ahora. También podemos agrupar registros por un criterio determinado, como por ejemplo, agrupar por clientes las ventas realizadas.

Cada grupo tendrá como resultado de la consulta una fila resumen que contiene la información del grupo.

Para la formación de grupos adicionamos, a la forma básica de la sentencia *SELECT* vista anteriormente, la orden *GROUP BY* ubicada antes de *ORDER BY*, como se muestra a continuación:

```
SELECT    Lista...
FROM      Tabla, Tabla...
WHERE     Condiciones
GROUP BY  Expresión, Expresión,...
          ORDER BY Expresión, Expresión,...
;         Terminador
```

Las funciones para el procesamiento de grupos son:

COUNT(columna)	Cantidad de registros en que la columna tiene valores no nulos.
COUNT(*)	Cantidad de registros que hay en la tabla, incluyendo los valores nulos.
MIN(columna)	Valor mínimo del grupo.
MAX(columna)	Valor máximo del grupo.

SUM(columna) Suma los valores del grupo.

AVG(columna) Calcula valor medio del grupo, sin considerar los valores nulos.

La lista de columnas a mostrar en la consulta puede contener las funciones de grupo, así como la columna o expresión usada para formar los grupos en la orden *GROUP BY*. En una misma consulta no se pueden mezclar funciones de grupo con columnas o funciones que trabajan con registros individuales.

Las ventas por cliente es un buen ejemplo para mostrar el uso de los grupos. En el siguiente caso se hace un resumen de ventas por cliente, con la cantidad de ventas, valor mínimo, medio y máximo, así como la suma total de ventas. La formación del grupo será por el nombre del cliente y la columna a cuantificar para cada grupo será el valor de las ventas.

```
SELECT nombre "CLIENTE",
     COUNT(valor) "VENTAS",
     MIN(valor) "MINIMA",
     AVG(valor) "MEDIA",
     MAX(valor) "MAXIMA",
     SUM(VALOR) "TOTAL"
   FROM clientes,ventas
   WHERE clientes.codigo=ventas.codigo
   GROUP BY nombre
;
```

CLIENTE	VENTAS	MINIMA	MEDIA	MAXIMA	TOTAL
CASA AUGE DEPORTES	3	33.0	138.667	330.0	416.0
CLUB DE TENNIS	3	100.5	237.167	500.5	711.5
FERIA DEL LIBRO	3	110.0	203.333	310.0	610.0
PINTURERIAS PROPIOS	3	20.5	90.500	220.5	271.5

Tabla 2: Estadística de la base de datos

Como se puede observar en la tabla 2, a cada columna se le asignó un título de cabecera, que se escribe entre comillas dobles a la derecha de la columna, con el objetivo de mejorar la presentación de la consulta.

El orden en las consultas por grupos, cuando no esta presente la orden *ORDER BY*, está dado por la columna que forma los grupos. Si deseamos cambiar ese orden, como es el caso de ordenar por el valor total de ventas, se debe adicionar al final la orden *ORDER BY SUM(VALOR).*

En el ejemplo del fuente 6 se forman grupos por artículo para obtener la cantidad de ventas, valor mínimo, medio y máximo, así como la suma total de ventas para cada artículo, ordenado de mayor a menor por la venta total.

```
Fuente 6

SELECT articulo, COUNT(valor) "VENTAS",MIN(valor) "MINIMA",
    AVG(valor) "MEDIA",MAX(valor) "MAXIMA",SUM(VALOR) "TOTAL"
    FROM ventas
    GROUP BY articulo
    ORDER BY SUM(valor) DESC
;
```

ARTICULO	VENTAS	MINIMA	MEDIA	MAXIMA	TOTAL
PAPEL	8	100.5	234.00	500.5	1872
DISKETTE	4	20.5	34.25	53.0	137

Tabla 3: Otra estadística de la base de datos

Subconsultas

Otro aspecto de fácil diseño y uso que muestra una vez más las posibilidades de SQL son las subconsultas.

Subconsulta es aquella consulta de cuyo resultado depende otra consulta, llamada principal, y se define como una sentencia *SELECT* que esta incluida en la orden *WHERE* de la consulta principal. Una subconsulta, a su vez, puede contener otra subconsulta y así hasta un máximo de 16 niveles.

Las particularidades de las subconsultas son:

1. Su resultado no se visualiza, sino que se pasa a la consulta principal para su comprobación.

2. Puede devolver un valor único o una lista de valores y en dependencia de esto se debe usar el operador del tipo correspondiente.

3. No puede usar el operador *BETWEEN*, ni contener la orden *ORDER BY*.

4. Puede contener una sola columna, que es lo más común, o varias columnas. Este último caso se llama subconsulta con columnas múltiples. Cuando dos o más columnas serán comprobadas al mismo tiempo, deben encerrarse entre paréntesis.

Expliquemos como se construye una subconsulta con el siguiente ejemplo, donde necesitamos saber ¿cuál fue la mayor venta realizada?. Para ello, diseñemos una subconsulta que busque el valor máximo de venta con el uso de la función *MAX(valor)* y una consulta principal que muestre las ventas iguales al máximo valor encontrado por la subconsulta. Veamos el fuente 7.

```
Fuente 7

SELECT nombre,articulo,valor
FROM clientes,ventas
WHERE valor = ( Subconsulta para buscar
                SELECT MAX(valor)          el valor máximo de venta.
                FROM ventas
              )
      AND clientes.codigo=ventas.codigo
;
```

NOMBRE: CLUB DE TENNIS
ARTICULO: PAPEL
VALOR: 500.5

Otra aplicación clásica de la subconsulta es cuando deseamos saber las ventas de un artículo

realizadas por encima de su venta promedio. En este caso, es necesario realizar los pasos mostrados en el fuente 8:

```
Fuente 8

SELECT   nombre,valor "PAPEL"
FROM     clientes,ventas
WHERE    clientes.codigo=ventas.codigo
    AND articulo='PAPEL'
    AND valor >
                (                              Subconsulta de
                 SELECT AVG(valor)             venta promedio de
                 FROM   ventas                 papel.
                 WHERE  articulo='PAPEL'
                )
ORDER BY valor DESC            Ordenado por valor de venta
;                              en forma descendente
```

NOMBRE	PAPEL
CLUB DE TENNIS	500.5
CASA AUGE DEPORTES	330.0
FERIA DEL LIBRO	310.0

Tabla 4: ventas por encima de su venta promedio

Grupos con subconsulta

Hasta el momento estudiamos por separado un conjunto de herramientas de SQL, viendo en cada caso sus posibilidades. Ahora pasaremos a ver la combinación de grupos y subconsultas, lo que multiplica las posibilidades de SQL en cuanto al rendimiento en el diseño de consultas complejas se refiere, las cuales se pueden realizar en forma sencilla y con pocas líneas de código.

Para combinar grupos con subconsulta debemos incluir en la sentencia *SELECT* la orden *HAVING*, que tiene las siguientes características:

1. Funciona como la orden *WHERE*, pero sobre los resultados de las funciones de grupo, en oposición a las columnas o funciones para registros individuales que se seleccionan mediante la orden *WHERE*. O sea, trabaja como si fuera una orden *WHERE*, pero sobre grupos de registros.

2. Se ubica después de la orden GROUP BY.

3. Puede usar una función de grupo diferente a la de la orden SELECT.

El ejemplo a diseñar para nuestra aplicación es la consulta ¿cuál fue el artículo más vendido y en qué cantidad?. En este caso, la orden *HAVING* de la consulta principal selecciona aquellos artículos (*GROUP BY*) que tienen una venta total (*SUM(valor)*) igual a la mayor venta realizada por artículo (*MAX(SUM(valor))*) que devuelve la subconsulta.

La sentencia *SELECT* y el resultado de nuestra consulta sería la del fuente 9 y la tabla 5:

```
Fuente 9
```

```
SELECT articulo "ARTICULO MAS VENDIDO",SUM(valor) "VENTA"
    FROM ventas
    GROUP BY articulo
    HAVING SUM(valor) =
    (                              Subconsulta para buscar
     SELECT MAX(SUM(valor))        el artículo más vendido
     FROM ventas                   con la formación de grupos
     GROUP BY articulo             por artículo.
    )
;
```

ARTICULO MÁS VENDIDO	VENTA
PAPEL	1872

Tabla 4: ventas por encima de su venta promedio

Indices

El índice es un instrumento que aumenta la velocidad de respuesta de la consulta, mejorando su rendimiento y optimizando su resultado. El manejo de los índices en *ORACLE* se realiza de forma *inteligente*, donde el programador sólo crea los índices sin tener que especificar, explícitamente, cuál es el índice que va a usar. Es el propio sistema, al analizar la condición de la consulta, quien decide qué índice se necesita. Por ejemplo cuando en una consulta se relacionan dos tablas por una columna, si ésta tiene definido un índice se activa, como en el caso cuando relacionamos la tabla de clientes y ventas por la columna código para identificar al cliente (*WHERE clientes.codigo=ventas.codigo*)

La identificación del índice a usar está relacionada con las columnas que participan en las condiciones de la orden *WHERE*. Si la columna que forma el índice está presente en alguna de las condiciones éste se activa. No obstante, existen casos en que la presencia de la columna no garantiza el uso de su índice, ya que éstos son ignorados, como en las siguientes situaciones cuando la columna indexada es:

Evaluada con el uso de los operadores *IS NULL* o *IS NOT NULL*.

```
SELECT nombre,articulo,valor
FROM   clientes,ventas
WHERE  nombre IS NOT NULL;
```

Modificada por alguna función, excepto por las funciones *MAX(columna)* o *MIN(columna).*

```
SELECT nombre,articulo,valor
FROM   clientes,ventas
WHERE UPPER(nombre)>' '
;
```

Usada en una comparación con el operador *LIKE* a un patrón de consulta que comienza con alguno de los signos especiales (*% _*).

```
SELECT nombre,articulo,valor
      FROM   clientes,ventas
      WHERE  nombre
      LIKE '%DEPORTE%'
```

```
;
```

Finalmente debemos aclarar que los registros cuyo valor es *NULL* para la columna indexada, no forman parte del índice. Por lo tanto, cuando el índice se activa estos registros no se muestran como resultado de la consulta.

Con lo estudiado en nuestros dos primeros artículos sobre *Oracle Básico*, aprendimos a crear nuestras tablas e índices y a construir las consultas. Por lo tanto, ya tenemos los elementos necesario para analizar las particularidades del *Diseño de Pantallas* con el módulo *SQLFORMS*, con el objetivo de ingresar, modificar, eliminar y consultar las tablas en forma amigable y en la medida de las necesidades del usuario. A este tema dedicaremos nuestro próximo artículo.

Oracle básico (III): Diseño de pantallas con *SQLForms*

SQLForms es la herramienta de *Oracle* que permite, de un modo sencillo y eficiente, diseñar pantallas para el ingreso, modificaciones, bajas y consultas de registros. El usuario podrá, una vez definida la forma, trabajar con ella sin necesidad de generar códigos, dado que Oracle trae incorporado un conjunto de procedimientos y funciones asociados a las teclas de funciones, como por ejemplo la tecla [F7], que se usa para iniciar una consulta.

El objetivo de este artículo es el estudio de los conceptos básicos de *SQLForms*, a partir de los cuales el lector estará en condiciones de profundizar independientemente con el la documentación existente sobre Oracle, que es completa, voluminosa y con ejemplos muy ilustrativos.

Forma

La forma elegida para el diseño es la de *Cliente-Ventas*, cuyo objetivo, como se muestra en la siguiente figura, es mostrar los datos básicos del cliente y las ventas realizadas:
Las tablas 1 y 2 representan la Forma *Cliente-Ventas*

Código	Fecha	Nombre	Teléfono	Dirección	Anotación
3	27-09-95	Feria del libro	234555	Canelones 1800	

Tabla 1: Bloque Cliente

Fecha	Artículo	Valor
20/09/95	Papel Fanfold	110
11/09/95	Disquete	190
24/08/95	Papel Fotocopia	310

Tabla 2: Bloque Ventas

La forma se organiza en bloques de información, donde cada uno tiene asociado una tabla de datos y las columnas seleccionadas. La forma puede ocupar una o varias pantallas. En el ejemplo, como se puede observar, ocupa una pantalla.

Bloque

En nuestro ejemplo la forma está compuesta por dos bloques: *Cliente* y *Ventas*. A continuación damos la descripción de cada uno de ello, con su correspondiente definición en *SQLForms*.

Cliente

Objetivo: Ficha básica con datos del cliente.
Tabla: CLIENTES.
Registros: Presentación simple, un registro por cliente.
Tipo: Bloque Principal (*Master Block*).
Orden: Por Nombre del cliente (*ORDER BY NOMBRE*).

Pantalla de definición:

Block: CLIENTE	Records	Array Size:
Table: CLIENTES	Displayed: 1	[ ] Prim Key
Sequence Number: 1	Buffered:	[ ] In Menu
	Lines per:	[ ] Column Sec
Default Where/Order By:	ORDER BY NOMBRE	

Ventas

Objetivo:	Ventas realizadas a un cliente.
Tabla:	VENTAS.
Registros:	Presentación Múltiple, varios registros por cliente, donde cada registro ocupa una línea.
Tipo:	Bloque Detalle, cuya información detalla las ventas del cliente representado en el bloque Principal. La relación entre bloques puede establecerse por uno o más campos. En este caso el campo CODIGO del cliente es el que relaciona ambos bloques. Por eso definimos como condición de relación CLIENTE.CODIGO = VENTAS.CODIGO.
Orden:	Por fecha de venta en forma descendiente (ORDER BY FECHA DESC).

Pantalla de definición:

Block: VENTAS	Records	Array Size: 3
Table: VENTAS	Displayed: 5	[ ] Prim Key
Sequence Number: 2	Buffered: 5	[ ] In Menu
	Lines per: 1	[ ] Column Sec
Default Where/Order By:	ORDER BY FECHA DESC	
Master Block: CLIENTE	[X] Delete Details	
Join Condition	CLIENTE.CODIGO = VENTAS.CODIGO	

Campo

Los datos de la forma se llaman campos, pudiendo los mismos representar columnas de la tabla o variables de memoria. La identificación del campo está compuesta por el nombre del bloque y el nombre del campo, como por ejemplo :*CLIENTE.CODIGO* y :*VENTAS.CODIGO*.

En el momento de crear la forma se determinan para cada campo:

Definiciones básicas

Nombre.
Nº de orden.
Tipo de dato.
Ancho del campo, consulta y visualización.
Posición en pantalla.

Definiciones avanzadas

Formato de presentación. El formato de presentación del campo tipo fecha es *DD-MON-YY*, pudiendo ser cambiado al formato dd-mm-yy o dd/mm/yy.

Valores por defecto: Si en el momento del alta de clientes se desea generar una codificación numérica secuencial en forma automática, se debe asignar al campo *CODIGO* del bloque *CLIENTE* el siguiente valor de la secuencia CODIGO_CLIENTE, previamente creada en

SQLPLUS. En este caso se define el valor por defecto como *SEQUENCE.CODIGO_CLIENTE.NEXTVAL*

A un campo de tipo fecha que se desee iniciar con la fecha del sistema se le asigna un valor por defecto igual a $$date$$, que es la variable del sistema que contiene la fecha.

Rango de valores: Para asignar un rango de valores a un campo se definen sus valores extremos.

Campo de relación: Define si el campo esta relacionado a un campo del bloque principal. En nuestro ejemplo el campo :VENTAS.CODIGO se relaciona con el campo :CLIENTE.CODIGO del bloque principal.

Lista de valores: Asigna a un campo una lista de valores a consultar. Para consultar la lista de clientes por nombre o por código, se define una lista de valores para el campo :CLIENTE.CODIGO, como se muestra a continuación:

Titulo: CLIENTES
Posición: X: 10 Y: 10
Comando SQL: SELECT NOMBRE,CODIGO
INTO :CLIENTE.NOMBRE,:CLIENTE.CODIGO
FROM CLIENTES ORDER BY NOMBRE

En la siguiente figura se muestran las pantallas con las definiciones básicas y avanzadas de los campos: *CLIENTE.CODIGO* y :*VENTAS.CODIGO*

:Cliente.Codigo

Field Name: CODIGO
Sequence Number: 1
Data Type: NUMBER (Select Attributes)
Field Length: 10
Query Length: 10
Display Length: 10
Screen Position: X: 20 Y: 4
Page: 1 (Editor Attributes)
Format Mask:
Default Value: SEQUENCE.CODIGO_CLIENTE.NEXTVAL
Hint: Enter value for : CODIGO
Valid Range: Low: High:
Enforce Key:
List of Values: Title: CLIENTES
Pos: X: 10 Y: 10
List of Values SQL Text:

```
SELECT NOMBRE,CODIGO
      INTO :NOMBRE,:CODIGO
      FROM CLIENTES
      ORDER BY NOMBRE
```

:Ventas.codigo

Field Name: CODIGO
Sequence Number: 4
Data Type: NUMBER (Select Attributes)

Field Length:	10
Query Length:	10
Display Length:	10
Screen Position:	X: Y:
Page:	(Editor Attributes)
Format Mask:	
Default Value:	
Hint:	
Valid Range:	Low: High:
Enforce Key:	**CLIENTE.CODIGO**
List of Values:	Title:
Pos:	X: Y:

Atributos

Los atributos definen las siguientes características de un campo:

Tipo de campo:	Columna de tabla o Variable de memoria.
Clave Primaria:	Indica que los registros ingresados tienen un único valor en este campo.
Mostrar:	Muestra el valor del campo.
Obligatorio:	Se requiere ingresar un valor. No puede ser NULL.
Ingresar:	Se puede ingresar información en el campo.
Modificar:	Indica que se puede cambiar el valor del campo después de realizar una consulta.
Modificar si es Null:	Indica que se puede cambiar el valor del campo después de realizar una consulta, solamente en el caso que el valor del campo sea NULL.
Consultar:	Indica que en modo consulta se puede escribir una condición de consulta.
Conversión a Mayúscula:	Conversión automática a mayúscula.
Visualizar:	Visualizar el contenido del campo al momento de ingresar información. En caso contrario se muestra en blanco.
Ancho Fijo:	Indica que el valor a ingresar tiene que ser del ancho del campo.
Salto automático:	Cuando se llena el campo pasa automáticamente al siguiente.
Texto de Ayuda:	Se muestra al momento de ingresar el campo un texto de ayuda.

A continuación se muestra la relación de elementos que perfilan la definición de atributos para el campo *CLIENTE.CODIGO*

[X] Base Table
[X] Primary Key
[X] Displayed
[X] Required
[X] Input Allowed
[X] Update Allowed
[ ] Update if Null
[X] Query Allowed
[X] Uppercase
[X] Echo Input
[ ] Fixed Length
[ ] Automatic Skip
[ ] Automatic Hint

Consultas

Hasta el momento estudiamos la creación de formas. Ahora veremos cómo, sin necesidad de generar ningún código, ya estamos en condiciones de almacenar información y realizar las consultas correspondientes.

Antes de pasar a ver los distintos tipo de consulta, queremos detallar los pasos necesarios para habilitar una consulta:

- Iniciar la forma.
- Ir al bloque a consultar.
- Dar inicio a la consulta (con la tecla [F7]).
- Ubicar el cursor en el campo a consultar.
- Escribir la condición de consulta.
- Realizar la consulta (con la tecla [F8]).
- Ver el resultado de la consulta en pantalla.

A continuación estudiaremos los diferentes tipos de consultas, con un ejemplo para cada caso, estas consultas son:

- Exacta.
- Condicional.
- Aproximada.
- Múltiple.
- Avanzada.

Exacta

La consulta exacta es la que verifica una condición con el operador de *igual a*. Ilustremos este caso con la consulta de las ventas de un determinado artículo como puede ser *PAPEL FANFOLD*.

Los parámetros de consulta para *CLIENTE* serían:

CODIGO	1
FECHA	10-04-95
NOMBRE	PINTURERIAS PROPIOS
TELEFONO	45 67 89
DIRECCION	Uruguay 1234
ANOTACION	Ferretería y artículos para el Hogar

Y los parámetros correspondientes a *VENTAS*.

FECHA:	
ARTICULO:	**PAPEL FANFOLD**
VALOR:	

Condicional

La consulta condicional es la que incluye algún operador de comparación como <, <=, >, >=, !=.

Para consultar las ventas de cualquier artículo cuyo valor de venta esté por encima de 100, se define la siguiente consulta para *Clientes* y *Ventas*:

CODIGO	1
FECHA	10-04-95
NOMBRE	PINTURERIAS PROPIOS

TELEFONO 45 67 89
DIRECCION Uruguay 1234
ANOTACION Ferretería y artículos para el Hogar

FECHA:
ARTICULO:
VALOR: **>100**

Aproximada

La consulta aproximada es aquella que tiene un patrón de consulta a partir del cual *SQLForms* construye una condición con el operador *LIKE* (tema ya expuesto en el artículo *Oracle Básico (II)*).

Si necesitáramos consultar las ventas de cualquier tipo de papel para un cliente dado, bastaría con especificar el patrón *PAPEL%.* De esta forma serían consultados todos los artículos cuyo nombre comenzara con *PAPEL*, como se muestra a continuación:

CODIGO 1
FECHA 10-04-95
NOMBRE PINTURERIAS PROPIOS
TELEFONO 45 67 89
DIRECCION Uruguay 1234
ANOTACION Ferretería y artículos para el Hogar

FECHA:
ARTICULO: **PAPEL%**
VALOR:

Múltiple

Consulta múltiple es aquélla en la que participan varios campos en la condición a verificar. Por ejemplo, si necesitáramos consultar las ventas de cualquier tipo de papel cuyo valor este por encima de 100 pesos, realizaríamos la siguiente consulta:

CODIGO 1
FECHA 10-04-95
NOMBRE PINTURERIAS PROPIOS
TELEFONO 45 67 89
DIRECCION Uruguay 1234
ANOTACION Ferretería y artículos para el Hogar

FECHA:
ARTICULO: **PAPEL%**
VALOR: **>100**

Avanzada

Consulta avanzada es aquélla que combina diferentes condiciones para un mismo campo, o la que brinda la posibilidad de modificar el orden de presentación de los registros.

En este tipo de consulta se puede construir una condición con todas las posibilidades del mandato *SELECT* (estudiadas en el artículo *Oracle Básico (II)*), excepto el manejo de grupos con *GROUP BY*.

Veamos el siguiente ejemplo donde necesitamos buscar un cliente, cuyo nombre es *LABORATORIO CRUZ DEL SUR*, pero no se recuerda si fue registrado de forma completa o abreviada, como podría ser *L. CRUZ DEL SUR o LAB. CRUZ DEL SUR*.

Nuestra condición de consulta sería:

```
WHERE :NOMBRE LIKE 'L%' and :NOMBRE LIKE '%SUR%'
```

donde se buscan los clientes cuyo nombre comiencen con L y contienen la palabra SUR en cualquier lugar.

La pantalla de nuestra consulta sería:

CODIGO:
FECHA:
NOMBRE: **N**

Criteria: **:N LIKE 'L%' AND :N LIKE '%SUR%'**

En tipo de consulta, una vez que el cursor esta ubicado en el campo a consultar, a diferencia de los otros tipos de consultas debe realizar las siguientes acciones:

- Escribir un nombre de variable, como por ejemplo *:N*, que represente al campo *NOMBRE* en el criterio de consulta. Debe comenzarse con dos puntos para indicar que se está haciendo referencia a una variable y no a un valor de consulta.

- Con la tecla de realizar consulta, [F8] en nuestro caso, se habilita el cuadro donde se escribe el criterio de la consulta.

- Realizar la Consulta con [F10].

A continuación veamos cómo usar la consulta avanzada para cambiar el orden de presentación de los registros. En nuestra forma los registros del bloque *CLIENTE*, por definición, se presentan ordenados por nombre.

En el siguiente ejemplo deseamos consultar aquellos clientes registrado desde el 01/10/95 a la fecha de hoy y cuyo nombre contenga la palabra *LIBRO*, y el resultado debe estar ordenado en forma descendente por la fecha del registro.

CODIGO
FECHA **:F**
NOMBRE **:N**

Criteria:

```
:F >= TO_DATE('01/10/95','DD/MM/YY')
      AND :N LIKE '%LIBRO%'
      ORDER BY FECHA DESC
```

Hasta aquí estudiamos la creación de la forma y sus usos, sin aún pasar a la generación de códigos, tema de nuestro próximo artículo. Para ello, en la próxima entrega nos detendremos en tareas de programación con el lenguaje *PL/SQL* para crear disparadores (Trigger) y procedimientos que nos permitan automatizar determinadas tareas, como pueden ser:

- La coordinación de consulta entre el bloque principal *CLIENTE* y el bloque de detalle *VENTAS*.

- La validación de las modificaciones (*COMMIT*).

Oracle básico (IV): Programación en *PL/SQL*

El lenguaje de programación de *Oracle*, llamado *PL/SQL*, es un lenguaje portable, procedural y de transacción muy potente y de fácil manejo, con las siguientes características fundamentales:

1. Incluye todos los comandos de *SQL* estudiados en el artículo *Oracle Básico I y II* (ver revista ***Algoritmo*** números 8 y 9, respectivamente):

   - SELECT
   - INSERT
   - UPDATE
   - DELETE.

2. Es una extensión de SQL, ya que este es un lenguaje no completo dado que no incluye las herramientas clásicas de programación. Por eso, PL/SQL amplia sus posibilidades al incorporar las siguientes sentencias:

   - Control condicional

```
IF ... THEN ... ELSE ... ENDIF
```

   - Ciclos

```
FOR ... LOOP
WHILE ... LOOP
```

3. Incorpora opciones avanzadas en:

- Control y tratamiento de errores llamado excepciones.
- Manejo de cursores.
- Variedad de procedimientos y funciones empaquetadas incorporadas en el módulo *SQL*Forms* para la programación de disparadores (*Trigger*) y procedimientos del usuario (*Procedure*).

Estructura del bloque de código

Veamos a continuación la organización del bloque de código de *PL/SQL*, compuesto por cuatro secciones *DECLARE*, *BEGIN*, *EXCEPTION y END* como se detalla en el fuente 1:

```
/* --- Fuente 1 ------------------------------------------------------------

[<< nombre del bloque >>]
    Etiqueta que identifica al Bloque.

[DECLARE]
    Declaración de
        Variable
        Constante       Se inicializa con un valor que no se puede
                        modificar.
```

```
        Cursor          Area de trabajo que contiene los datos de la fila de
                        la tabla en uso. El cursor es el resultado de una
                        sentencia SELECT.
    ExcepciónVariables  para control de errores.
BEGIN
    Código.
[EXCEPTION]
    Control y tratamiento de errores.
    Es el punto al que se transfiere el control del programa siempre que
    exista un problema. Los indicadores de excepción pueden ser definidos por
    el usuario o por el sistema, como es por ejemplo la excepción
    ZERO_DIVIDE. Las excepciones se activan automáticamente al ocurrir un
    error, existiendo la definición de la excepción OTHERS que considera
    aquellos errores no definidos y que siempre se ubica al final de todas
    las excepciones.

END [nombre del bloque];
    Fin del Bloque.
```

Con el ejemplo del fuente 2 ilustraremos las distintas secciones que componen un bloque de código en *PL/SQL*. En este caso deseamos calcular la venta promedio del día y, en caso que la misma sea menor a lo esperado, se debe registrar en la tabla *VENTABAJA*.

```
/* --- Fuente 2 ------------------------------------------------------------

DECLARE
    esperada          CONSTANT NUMBER(5) := 500;
    xtotal            NUMBER;
    xcant             NUMBER;
    xprom             NUMBER;
BEGIN
    /*Asigna a la variable xtotal el TOTAL de las ventas
    y a la variable xcant la cantidad de ventas del día.
    */
    SELECT SUM(valor),COUNT(valor) INTO xtotal,xcant
        FROM ventas WHERE fecha=sysdate;
    xprom:=xtotal/xcant;
    IF xprom >= esperada THEN
        message('Ventas por encima de la esperada');
        pause;
    ELSE
        /*Se registra en la tabla ventabaja las ventas por debajo
        del promedio esperado */
        INSERT INTO ventabaja VALUES (sysdate,xprom);
    END IF;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        message('No se realizaron ventas en el día');
    pause;
    WHEN OTHERS THEN
        message('Error Indefinido');
    pause;
END;
```

Asignación de valores

Las dos formas que existen para asignar valores a variables de memoria, vistas en el ejemplo anterior, son:

- Con el operador de asignación :=, como cuando calculamos el promedio de las ventas

asignándole valor a la variable xprom con la siguiente sentencia:

```
xprom:=xtotal/xcant;
```

- Con la sentencia SELECT que contiene la orden INTO, como se muestra, es la asignación de valores a las variables xtotal y xcant con el siguiente código:

```
SELECT SUM(valor),
    COUNT(valor)
    INTO xtotal,xcant
    FROM ventas
    WHERE fecha=sysdate;
```

Veamos a continuación, con la creación del procedimiento *FECHAALTA*, la asignación de valores a una variable de registro llamada *Client_Rec*, que va a contener la estructura de una fila de la tabla *CLIENTES* y que estará formada por todos los campos correspondientes a la tabla. Para esto usaremos el atributo de variable *%ROWTYPE* que declara una variable de registro que contiene la estructura de la tabla, y después, con el uso de la sentencia *SELECT * INTO,* se asigna a la variable de registro los valores de la fila. La referencia a un dato contenido en la variable de registro se hace de la forma *variable_registro.campo*, como por ejemplo *cliente_rec.fecha* hace referencia a la fecha del alta del cliente.

Pasemos a mostrar lo anteriormente expuesto a través del código del fuente 3.

```
/* --- Fuente 3 ------------------------------------------------------------

PROCEDURE FECHAALTA IS
    BEGIN
      DECLARE
          cliente_rec  clientes%ROWTYPE;
      BEGIN
          SELECT * INTO cliente_rec
              FROM clientes
              WHERE codigo = 5;
          IF cliente_rec.fecha>sysdate-10
          THEN
              message( cliente_rec.nombre||
                       ' Dado de alta en los últimos 10 días');
              pause;
          ELSE
              message( cliente_rec.nombre||
                       ' Dado de alta hace más de 10 días');
              pause;
          END IF;
    END;
END;
```

SELECT con control de excepciones

La sentencia *SELECT* en *PL/SQL* no muestra en pantalla las filas resultantes de la consulta, como ocurre en *SQL* (el cual trabaja en forma interactiva) sino que, según sea la acción a realizar, así será la cantidad de filas devueltas por la consulta, existiendo en este caso una de las tres posibles situaciones recogidas en la tabla 1:

Cantidad de filas	Acción
Una	Se realiza la siguiente sentencia
Más de una	Ocurre la excepción TOO_MANY_ROWS
Ninguna	Ocurre la excepción NO_DATA_FOUND

Tabla 1: Situaciones posibles según la búsqueda realizada

Por esta razón, veremos a continuación, a través de un ejemplo, el uso de la sentencia *SELECT*, con control de excepciones para definir la acción a realizar en dependencia de la cantidad de filas devueltas por la consulta.

Veamos con el código del fuente 4 en el que se define el procedimiento *VentasDe* para consultar las ventas realizadas en el día de un determinado artículo:

```
/* --- Fuente 4 ------------------------------------------------------------

PROCEDURE ventasde(xarticulo ventas.articulo%TYPE) is
    BEGIN
       DECLARE xnombre  clientes.nombre%TYPE;
           xventas      NUMBER;
    BEGIN
       SELECT  nombre into xnombre
       FROM    clientes,ventas
       WHERE   clientes.codigo=ventas.codigo
           AND
               ventas.fecha=sysdate
           AND
               articulo=xarticulo;
               message( 'Solo una venta de '
                        xarticulo¦¦' a: '¦¦xnombre);
    EXCEPTION
       WHEN NO_DATA_FOUND THEN
           message('No hay ventas de '||xarticulo);
           pause;
       WHEN  TOO_MANY_ROWS THEN
           SELECT   COUNT(*) INTO XVENTAS
           FROM     ventas
           WHERE    ventas.fecha=sysdate
               AND
               articulo=xarticulo;
               message( TO_CHAR(xventas)¦¦
                        ' Ventas de '¦¦xarticulo);
               pause;
       WHEN OTHERS THEN
           message('Error Indefinido');
           pause;
    END;
END;
```

Este procedimiento *ventasde* recibe un parámetro que es el nombre del artículo a consultar, por lo cual, para ejecutarlo, se debe escribir *ventasde('PAPEL')*.

Obsérvese también que la sentencia *SELECT COUNT(*) INTO*, usada en este ejemplo, siempre devuelve una fila, ya que no existe formación de grupos al no estar presente la orden *GROUP BY*. Por lo tanto, en este caso la única acción posible a realizar es pasar a la siguiente sentencia, o sea, no se requiere control de excepciones.

Manejo de cursores

El conjunto de filas resultantes de una consulta con la sentencia *SELECT*, como vimos anteriormente, puede estar compuesto por ninguna, una o varias filas, dependiendo de la condición que define la consulta. Para poder procesar individualmente cada fila de la consulta debemos definir un cursor (que es un área de trabajo de memoria) que contiene los datos de las filas de la tabla consultada por la sentencia *SELECT*.

Los pasos para el manejo de cursores, tema novedoso en la programación de *Oracle* con *PL/SQL*, son:

- Definir el cursor, especificando la lista de parámetros con sus correspondientes tipos de datos y estableciendo la consulta a realizar con la sentencia *SELECT*.
- Abrir el cursor para inicializarlo, siendo éste el momento en que se realiza la consulta.
- Leer una fila del cursor, pasando sus datos a las variables locales definidas a tal efecto.
- Repetir el proceso fila a fila hasta llegar a la última.
- Cerrar el cursor una vez que se terminó de procesar su última fila.

A continuación veremos un ejemplo de cursor con las siguientes características:

Objetivo: Consultar las ventas de una fecha dada ordenadas de mayor a menor.

Nombre: *CVENTAS*.

Parámetros: cfecha, variable que contiene la fecha a consultar.

Código de definición del cursor: Ver figura 1

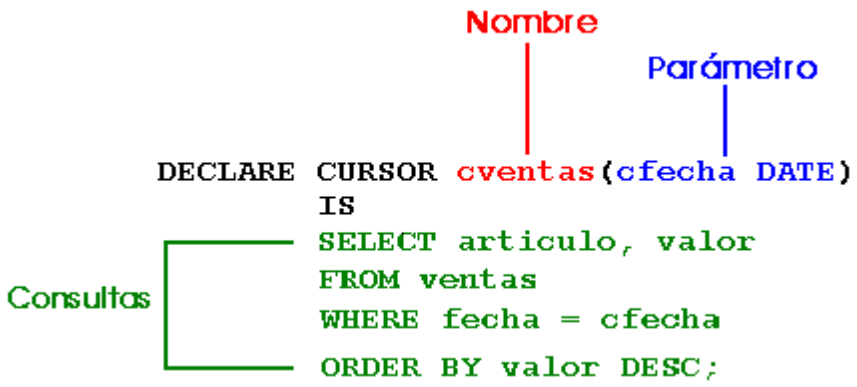

Figura 1: Código de definición de cursor

Con el procedimiento *VENTAS5* del fuente 5, mostraremos cómo usar el cursor *cventa* anteriormente definido, con el fin de registrar en la tabla *VENTAMAYOR* las 5 mayores ventas en una fecha dada.

```
/* --- Fuente 5 ------------------------------------------------------
```

```
PROCEDURE VENTAS5 (xfecha DATE) is
BEGIN

    DECLARE CURSOR cventas (cfecha DATE)
       IS SELECT articulo,valor
           FROM ventas
           WHERE fecha=cfecha
           ORDER BY valor DESC;

       xarticulo   ventas.articulo%TYPE;

       xvalor      ventas.valor%TYPE;

    BEGIN

       OPEN cventas(xfecha);

       FOR i IN 1..5 LOOP
           FETCH cventas INTO xarticulo,xvalor;
           EXIT WHEN cventas%NOTFOUND;
           INSERT INTO  ventamayor VALUES
                        (xfecha,xarticulo,xvalor);
           COMMIT;
       END LOOP;

       CLOSE cventas;
    END;
END;
```

Para llamar al procedimiento *ventas5* en una fecha dada, se puede escribir, por ejemplo:

```
ventas5(to_date('15/11/95','DD/MM/YY')
```

o

```
ventas5(sysdate).
```

A continuación detallaremos las sentencias usadas en este procedimiento:

```
DECLARE cursor
```

Define el cursor, su consulta y la lista de parámetros que se pasan a la orden *WHERE*, es solo la declaración del cursor y no la realización de la consulta.

```
xarticulo ventas.articulo%TYPE;
```

Define la variable *xarticulo* igual a la columna *articulo* de la tabla ventas, que con el uso del atributo de variable *%TYPE* permite declarar una variable del mismo tipo que una columna de la tabla. No es necesario conocer cómo está definida esa columna en la tabla y, en caso que la definición de la columna sea modificada, automáticamente se cambia la variable *xarticulo*.

```
OPEN cventas(xfecha);
```

Realiza la consulta asociada al cursor, pasando el valor del parámetro y guardando sus resultados en un área de la memoria, desde la cual, posteriormente, se pueden leer estas filas.

```
FOR i IN 1..5 LOOP
```

Ciclo numérico de repetición para poder consultar las 5 primeras ventas devueltas por el cursor.

```
FETCH cventas INTO xarticulo,xvalor;
```

Lee la siguiente fila de datos del cursor *cventas* y pasa los datos de la consulta a las variables *xarticulo* y *xvalor*.

```
EXIT WHEN cventas%NOTFOUND;
```

Garantiza la salida del ciclo antes de las última repetición, en caso que para una fecha dada se hayan efectuado menos de 5 ventas, ya que en esta situación la consulta del cursor devuelve menos de 5 filas.

%NOTFOUND es un atributo de cursor que es verdadero cuando la última sentencia *FETCH* no devuelve ninguna fila.

```
INSERT INTO ventamayor
   VALUES(xfecha,xarticulo,xvalor);
```

Insertar en la tabla *ventamayor* los valores leídos desde el cursor.

```
COMMIT;
```

Actualización de la tabla *ventamayor*.

```
END LOOP;
```

Fin del ciclo.

```
CLOSE cventas;
```

Cierra el cursor, eliminado sus datos del área de memoria.

Disparadores

El módulo *SQL*Forms* tiene incorporado una colección de procedimientos y funciones llamados "empaquetados" que se pueden incluir en el código de procedimientos o disparadores (*TRIGGER*) definidos por el usuario.

El disparador es un bloque de código que se activa cuando se pulsa una determinada tecla u ocurre cierto evento, como puede ser:

- Mover el cursor hacia o desde un campo, registro, bloque o forma.
- Realizar una consulta.
- Validar un dato.
- Hacer una transacción al insertar, modificar o eliminar registros de la base de datos.

Oracle asocia a cada tecla de función un procedimiento empaquetado, pudiendo el usuario redefinir esta asignación o capturar el disparador para ampliarlo o modificarlo con su propio código.

Usaremos como ejemplo la tecla [F9], que tiene asociado el disparador *KEY-LISTVAL*, que al activarse llama al procedimiento empaquetado *LIST_VALUES*, con el fin de consultar la lista de todos los valores del campo previamente codificado.

Ahora veamos cómo redefinimos el disparador *KEY-LISTVAL* con el objetivo de capturarlo para consultar, no sólo toda la lista de valores del campo, sino para incluir la potencia del operador *LIKE*, estudiados en el artículo *Oracle Básico (II): Consulta con SQL* (ver revista ***Algoritmo*** número 9), que nos permitirá consultar parte de la lista de valores (a partir de un patrón de consulta), brindando la posibilidad de incluir la consulta del tipo "comienzan con la palabra..." o de la consulta "contiene la palabra...".

En este caso el código sería el de la figura 2:

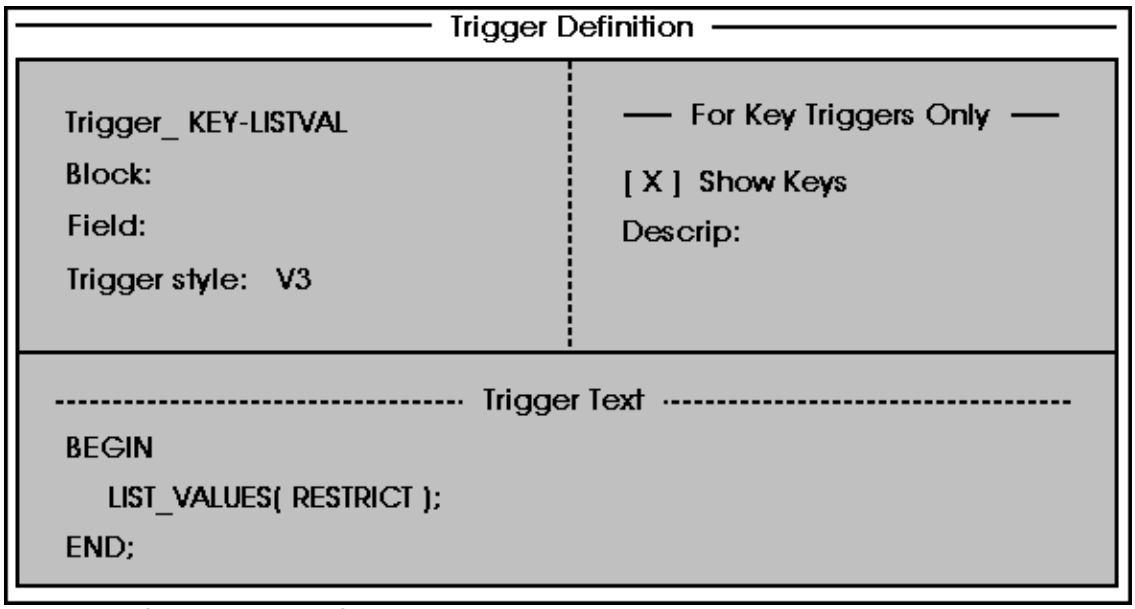

Figura 2: Código de definición del disparador (trigger)

Donde

KEY-LISTVAL Disparador que se activa con la tecla [F9] y que, por defecto, ejecuta el

procedimiento *LIST_VALUES* sin parámetros, que da la lista de todos los valores diferentes del campo.

LIST_VALUES (RESTRICT) Es el procedimiento empaquetado para obtener la lista de valores del campo, que con el parámetro *RESTRICT,* con la lógica del operador *LIKE,* permite ampliar las posibilidades de esta consulta; como explicamos anteriormente.

A partir de la versión 7 de *Oracle* el usuario puede almacenar, en forma independiente, sus funciones y procedimientos sin tener que escribirlos repetidamente para cada forma, y pudiendo compilarlos independientemente de las formas que lo usen. Pero, además, las funciones y procedimientos se pueden agrupar en un paquete para compartir definiciones, variables globales, constantes, cursores y excepciones, así como garantizar y revocar los permisos a nivel de paquete.

En el caso que sea necesario modificar el contenido del paquete, como el mismo se encuentra almacenado separadamente, no es necesario recompilar nada que use ese paquete, lo que facilita la gestión y mantenimiento de todos los procedimientos almacenados como una sola entidad para una determinada aplicación.

Además, en la versión 7, existe un nuevo tipo de disparador llamado *de base de datos*, que es un procedimiento asociado a una tabla que se activa cuando se produce un suceso que afecta a esa tabla. Su uso más común consiste en la definición de restricciones complejas de integridad.

Hasta aquí, he mostrado los conceptos básicos de la programación en *PL/SQL*, a partir de los cuales el lector estará en condiciones de seguir profundizando en el tema. Recomiendo usar el Manual de *Oracle: PL/SQL: Guía del Usuario*, que no sólo es detallado y claro, sino que también contiene una gran variedad de ejemplos.

En nuestro próximo artículo pasaremos a la creación de reportes (informes) con el módulo *SQL*Report* y al diseño de menús con *SQL*Menu*.

En principio me había propuesto tratar dos temas en esta entrega: reportes (informes) y menús. Pero, dada la extensión del primero de ellos, decidí dejar el diseño de menús para el siguiente artículo.

El módulo *SQLReport* de Oracle realiza de forma flexible, sencilla y eficiente la creación de reportes, informes o listados permitiendo, entre otras facilidades, la visualización previa por pantalla con una gran variedad en estilos de presentación.

Definiciones básicas

Para adentrarnos en el tema primero veremos las definiciones básicas, fundamento del diseño del reporte en Oracle:

Consulta

Define las columnas y filas de una o varias tablas que serán emitidas en el reporte, así como su orden de presentación. Una consulta puede estar subordinada a otra consulta principal, relacionadas por una o varias columnas.

Grupo

El grupo es una sección del reporte que representa al conjunto de columnas de la consulta, como una unidad, para determinar su ubicación en el reporte y su forma de presentación.

Para cada grupo se definen los siguientes atributos:

- Ubicación.
- Forma de presentación.
- Texto de cabecera y final.
- Título de las columnas.
- Ubicación de las columnas dentro del grupo.

En el momento de la definición de una consulta se crea, de forma automática, un grupo que contiene todos las columnas presentes en la lista de la sentencia *SELECT*. El nombre de este grupo se define como el nombre de la consulta, precedido por los caracteres *G_*. Por ejemplo, más adelante veremos como con la definición de la consulta *VENTAS* se genera, de forma automática, el grupo *G_VENTAS*.

Una consulta puede tener asociado más de un grupo, con el fin de separar las columnas de la consulta, por ejemplo para crear diferentes niveles en el reporte, como el cálculo de subtotales.

Campos

Cada columna de la consulta pasa a ser un campo del reporte. Además, podemos incluir como campos del reporte las siguientes variables:

- Del sistema como &DATE y &PAGE
- Del usuario, cuyo contenido es un comando SQL para realizar cálculos.

Parámetro

Variables definidas por el usuario para transferir datos a la consulta en el momento de la ejecución del reporte. Se hace referencia al parámetro en la consulta con su nombre precedido por dos puntos como *:nombre_parametro.*

Sumario

Define variables de tipo sumario para la realización de cálculos con el uso de las funciones *Sum*, *Min*, *Max*, *Count* y *Avg*.

Texto

Define la ubicación y forma de presentación de los campos dentro del grupo, así como los textos de inicio y final para cada grupo y para el reporte.

Para ubicar un campo o un parámetro en la sección de textos se hace referencia con su nombre, precedido con el carácter *&* de la forma: *&nombre*.

Primeros pasos

Veamos, a través del siguiente ejemplo, la creación del reporte *VENTAS*, cuyo objetivo es detallar las ventas realizadas en los últimos *n* días para un determinado cliente, identificado por su código y cuyo resultado debe ser presentado ordenado por artículo y fecha de venta.

Primero definimos la consulta *VENTAS* escribiendo la correspondiente sentencia *SELECT* como se muestra a continuación:

```
SELECT ARTICULO,FECHA,VALOR
FROM VENTAS
WHERE
       CODIGO=:XCOD
       AND FECHA>=SYSDATE-:XDIAS
ORDER BY ARTICULO,FECHA
```

Obsérvese que la sentencia *SELECT* de la consulta ***no*** debe terminar con punto y coma (;).

La consulta *VENTAS* debe recibir, en el momento de la realización del reporte, los siguientes parámetros:

- XCod para el código del cliente a consultar.
- XDias para la cantidad de días de las ventas.

Veamos a continuación la tabla 1, donde se da la definición de los parámetros:

Nombre del parámetro	Tipo	Ancho	Valor por defecto	Etiqueta
XCod	Num.	10	2	Código Cliente
XDias	Num.	5	15	Cantidad de días

Tabla 1: Definición de los parámetros

Desde mi punto de vista, el gran mérito del diseño de reporte en Oracle es su sencillez, ya que una vez realizadas las definiciones de consulta y parámetros, el reporte está pronto para ser emitido, gracias a que *SQLReport* se encarga del resto, incorporando, en forma automática, las siguientes definiciones:

1.- Creación del grupo *G_VENTAS* asociado a la consulta *VENTAS*.

Group Name	G_VENTAS
Query	VENTAS
Print Direction	Down

2.- Definición de los campos del reporte, tomados de la lista de la sentencia *SELECT* de la consulta *VENTAS*. Ver tabla 2.

Field Name	Source	Group	Label
ARTICULO	ARTICULO	G_VENTAS	Articulo
FECHA	FECHA	G_VENTAS	Fecha
VALOR	VALOR	G_VENTAS	Valor

Tabla 2: Campos del reporte

3.- Adiciona el texto de cabecera de cada columnas y la ubicación de las columnas dentro del grupo. Los elementos de la pantalla que tienen asignado algún valor son:

Lines Before:	1
Repeat On Page Overflow:	X
Justification:	Left

&ARTICULO
&FECHA
&VALOR

Ahora terminaremos el estudio de los elementos básicos del diseño de reportes viendo cómo emitir la identificación del cliente con su código, contenido en el parámetro *XCod*. Para esto incluiremos en el texto de cabecera del grupo del sistema *Report* (cuyo objetivo es representar parámetros y variables no asociados a ninguna consulta) la siguiente definición:

Object:	REPORT
Type:	Header
Status:	Edited
Justification:	Left
Cliente Nº:	&XCOD

En este momento nuestro reporte esta pronto para ser generado y emitido, para lo cual debemos elegir las tareas del menú *Generate* y *Execute*. Al momento de ejecutar el reporte, como se muestra en la tabla 3, se pedirá que se ingresen los valores de los parámetros:

Parameter	Value
Código Cliente	2
Cantidad de días	15

Tabla 3: Valores de los parámetros

El resultado de nuestro reporte *VENTAS* para el cliente 2 es el de la tabla 4.

Artículo	Fecha	Valor
DISKETTE	21/11/95	33
DISKETTE	21/11/95	100
PAPEL	22/11/95	150
PAPEL	27/11/95	53

Tabla 4: El resultado del reporte *VENTAS*

Mejorando nuestro reporte

Para ampliar y mejorar nuestro reporte, y con el fin de seguir avanzando en su diseño, estudiaremos cómo realizar las siguientes tareas:

- Imprimir la fecha del momento de emisión del reporte.
- Imprimir la fecha a partir de la cual se consultan las ventas.
- Incluir el nombre del cliente.

Comencemos por incluir la impresión de la fecha de emisión en la cabecera del reporte, para lo cual es necesario seguir los siguientes pasos:

1.- Crear el campo *EMISION* que contiene la variable del sistema *&DATE* que pertenece al grupo *REPORT*.

Field Name	EMISION
Source	&DATE
Group	REPORT
Label	Emision

2.- Modificar la cabecera del reporte para incluir el campo *EMISION* como se muestra a continuación (campos con datos, exclusivamente):

Object:	REPORT
Type:	Header
Status:	Edited
Justification:	Left
FECHA:	&EMISION
Cliente Nº:	&XCOD

Ahora, añadiremos en el reporte la fecha a partir de la cual se consultan las ventas, que se obtiene a partir de la fórmula *SYSDATE-:XDIAS*; para lo cual debemos realizar las siguientes tareas:

1.- Crear el campo *XDESDE* cuyo contenido es una sentencia *SELECT* del tipo *&SQL* para realizar el cálculo *SYSDATE-:XDIAS*. Este campo pertenece al grupo *REPORT*.

```
&SQL SELECT SYSDATE-:XDIAS
        INTO :XDESDE FROM DUAL¶
```

El comando *SQL*, asociado al campo *XDESDE*, es una sentencia *SELECT* con las siguientes características:

- Comienza con la palabra clave *&SQL*.
- Calcula una fórmula desde la tabla simbólica del sistema llamada *DUAL*.
- El resultado se pasa a la propia variable :*XDESDE* con el uso de la orden INTO.
- Sólo puede devolver una fila.

2.- Modificar la cabecera del reporte para incluir la fecha a partir de la cual se consultan las ventas, contenida en el campo *XDESDE*, como se muestra a continuación:

Object: REPORT
Type: Header
Status: Edited
Justification: Left
FECHA: &EMISION
Ventas desde: &XDESDE
Cliente Nº : &XCOD

Por último, veamos cómo incorporar el nombre del cliente junto a su código, con la realización de los siguientes pasos:

1.- Añadir una nueva consulta, llamada *CLIENTE*, con el fin de recuperar el nombre del cliente desde la tabla *CLIENTES,* como se muestra en la siguiente *SELECT*:

```
SELECT NOMBRE,CODIGO
      FROM CLIENTES
      WHERE CODIGO=:XCOD
```

2.- Modificar la consulta *VENTAS* para que pase a ser una consulta subordinada (subconsulta) a la consulta principal *CLIENTE,* relacionadas, además, por la columna *CODIGO* . Ver *SELECT* a continuación.

```
SELECT ARTICULO,FECHA,VALOR,CODIGO
    FROM VENTAS
    WHERE FECHA>=SYSDATE-:XDIAS
    ORDER BY ARTICULO,FECHA
```

Donde

Child Columns CODIGO
Parent 1 Columns CODIGO

3.- Ubicar el grupo *G_CLIENTE*, creado de forma automática por *SQLReport*, como primer grupo. Ver tabla 5.

Group name	Query	Print direction	Relative position
G_CLIENTE	CLIENTE	Down	

G_VENTAS	VENTAS	Down

Tabla 5: Ubicar el grupo *G_CLIENTE*

4.- Cambiar la posición del grupo *G_VENTAS* relativa a su grupo principal *G_CLIENTE*, que por defecto es a la derecha, a la posición debajo (*Below*). Ver tabla 6

Group name	Query	Print direction	Relative position
G_CLIENTE	CLIENTE	Down	
G_VENTAS	VENTAS	Down	Below

Tabla 6:Modificación de la posición relativa

5.- Definir el texto para el grupo *G_CLIENTE* como se muestra en la siguiente relación:

Object:	G_CLIENTE
Type:	Body
Status:	Edited
Repeat On Page Overflow:	X
Justification:	Left
Nombre:	&NOMBRE

La emisión del reporte después de las nuevas definiciones es el de la figura 1:

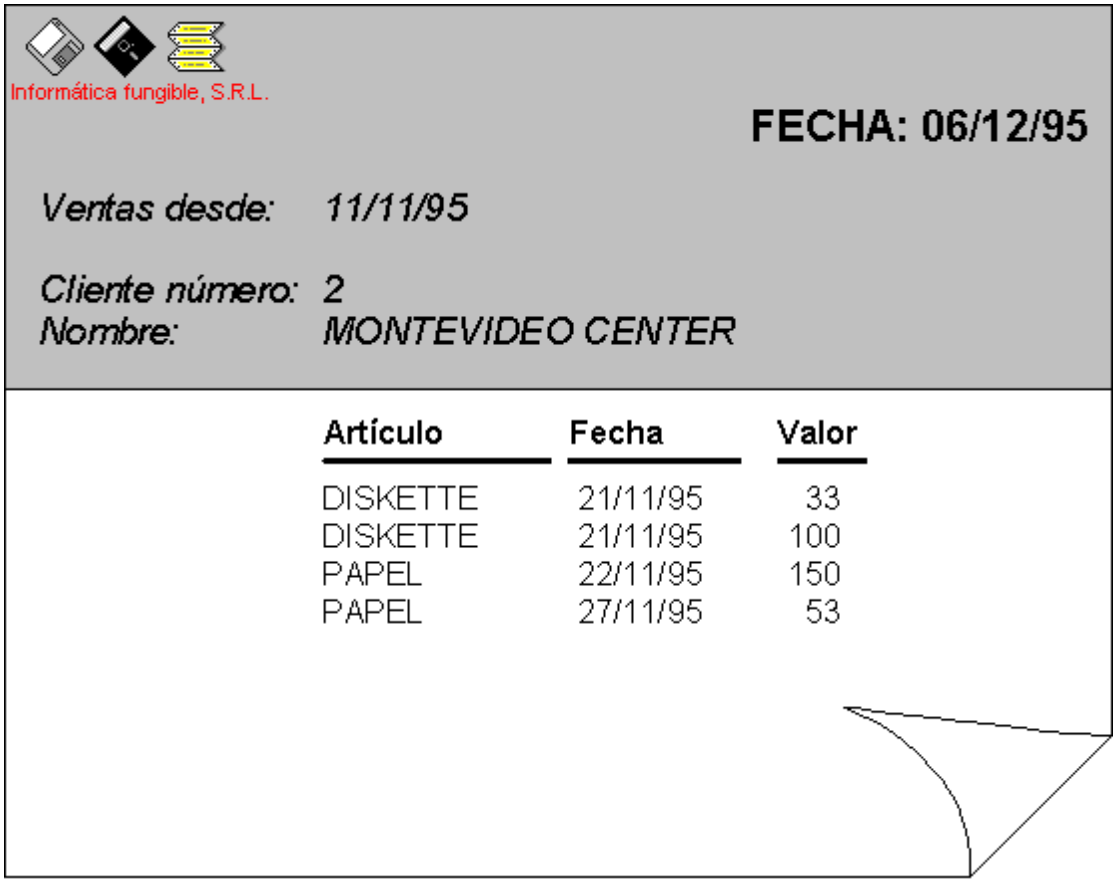

Figura 1: Ejemplo de reporte

Cálculos

Veamos a continuación cómo se incluyen cálculos en el reporte, para lo cual estudiaremos los siguientes casos:

- Cálculo de la cantidad de días de realización de cada venta.
- Venta total al cliente.
- Venta por artículo.

Empecemos por desarrollar el cálculo de la cantidad de días de realización de cada venta, para ello debemos llevar a cabo las siguientes tareas:

1.- Añadir en la consulta *VENTAS* la columna *DIAS* que contiene la fórmula *SYSDATE-FECHA*, como se muestra en la siguiente SELECT:

```
SELECT VALOR,ARTICULO,FECHA,
       CODIGO,SYSDATE-FECHA DIAS
       FROM VENTAS
       WHERE FECHA>=SYSDATE-:XDIAS
       ORDER BY ARTICULO,FECHA
```

2.- Añadir en el texto de cabecera de las columnas del grupo *G_VENTAS* el título para la nueva columna:

Object:	G_VENTAS
Type:	Column Heading
Status:	Edited
Repeat On Page Overflow:	X
Lines Before:	1
Justification:Left	
Valor:	Dias º

3.- Incluir el campo *DIAS* en el texto del grupo *G_VENTAS*.

Object:	G_VENTAS
Type:	Body
Status:	Edited
Repeat On Page Overflow:	X
Justification:	Left
&ARTICULO	
&FECHA	
&VALOR	
&DIAS	

Ahora veamos el cálculo de la venta total al cliente, con los siguientes pasos:

1.- Definir una variable de tipo sumario, llamada *TOTAL*, asociada al campo *VALOR* y cuya función es su suma, siendo *REPORT* el grupo, tanto de impresión como de cálculo. Ver tabla 7.

Summary name	Field	Funtion	Data type	Print group	Reset group

TOTAL	VALOR	Sum	NUM	REPORT	REPORT

Tabla 7: Variable *TOTAL* de tipo sumario

2.- Incluir el campo *TOTAL* en el texto final del reporte como se muestra en los datos siguientes:

Object: REPORT
Type: Footer
Status: Edited
Justification: Left
Spaces Before: 30
TOTAL
&TOTAL

Una vez terminadas las nuevas definiciones la emisión del reporte brindará el resultado que se ve en la figura 2:

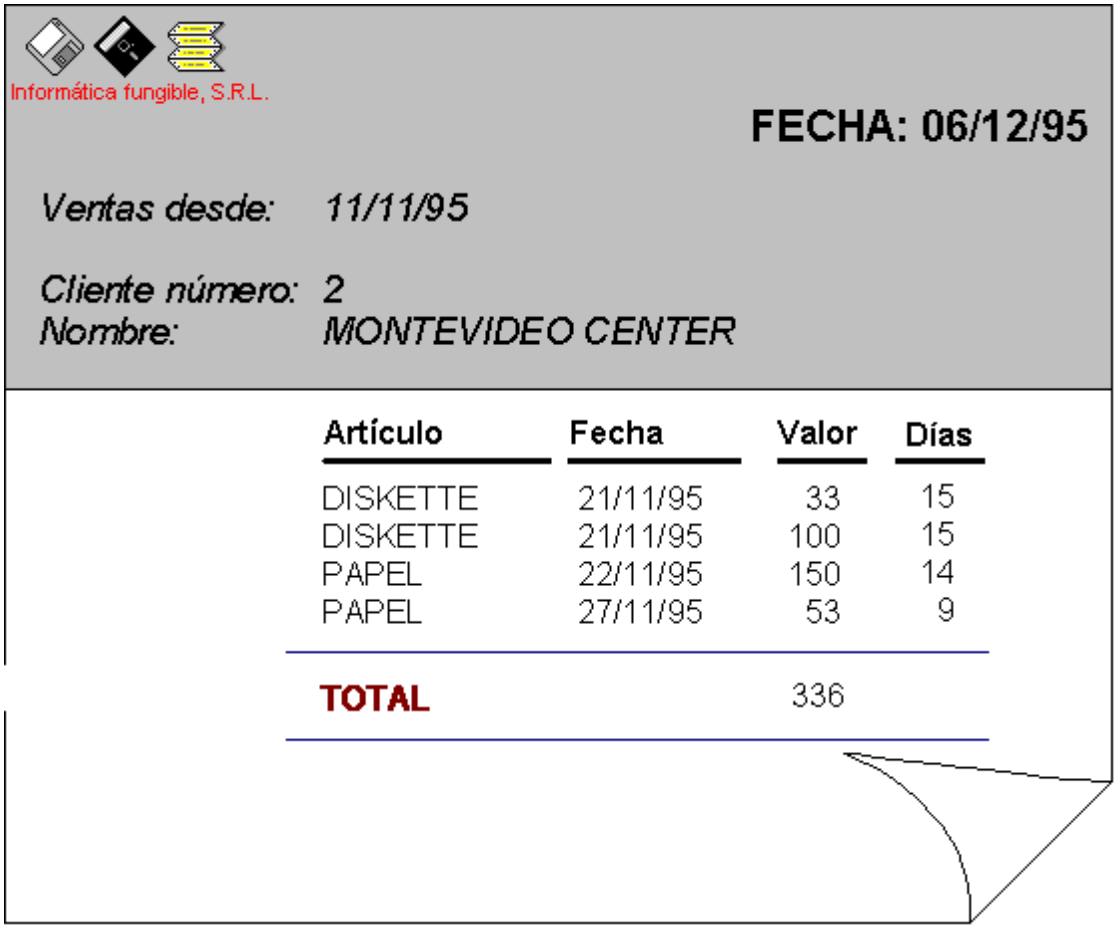

Figura 2: Ejemplo de reporte con total

Por último estudiaremos el cálculo de la venta por artículo (subtotales), para lo cual se necesita realizar las siguientes tareas:

1.- Definir un segundo grupo para la consulta *VENTAS*, llamado *G_ARTICULO*, que

identificará el campo, para el cual se van a calcular los subtotales que, en nuestro caso, es el campo *ARTICULO*. La ubicación de este grupo debe ser anterior al grupo *G_VENTAS*.

Es imprescindible que la consulta esté ordenada por el campo en que se van a calcular los subtotales. En nuestro caso la sentencia *SELECT* de la consulta *VENTAS* ya incluye la orden *ORDER BY ARTICULO*. Ver tabla 8.

Group name	Query	Print direction
G_CLIENTE	CLIENTE	Down
G_ARTICULO	VENTAS	Down
G_VENTAS	VENTAS	Down

Tabla 8: Modificación de la posición relativa

2.- Cambiar de grupo al campo *ARTICULO* pasándolo de *G_VENTAS* a *G_ARTICULO*.

Field Name ARTICULO
Source ARTICULO
Group G_ARTICULO
Label Articulo

3.- Definir la variable *SUBTOTAL*, de tipo sumario, cuyo fin es calcular los subtotales de las ventas, (efectuando la suma del campo *VALOR*), su grupo de cálculo es *G_ARTICULO* y el de impresión es *G_VENTAS*. Ver tabla 9.

Summary name	Field	Funtion	Data type	Print group	Reset group
SUBTOTAL	VALOR	Sum	NUM	G_VENTAS	G_ARTICULO

Tabla 9: Variable *SUBTOTAL* de tipo sumario

4.- Ubicar el campo *SUBTOTAL* en el texto final del grupo *G_VENTAS*.

Object: G_VENTAS
Type: Footer
Status: Edited
Justification: Left
SUBTOTAL
&SUBTOTAL

El resultado final de nuestro reporte *VENTAS,* con el cálculo de las ventas por artículo, es el de la figura 3.

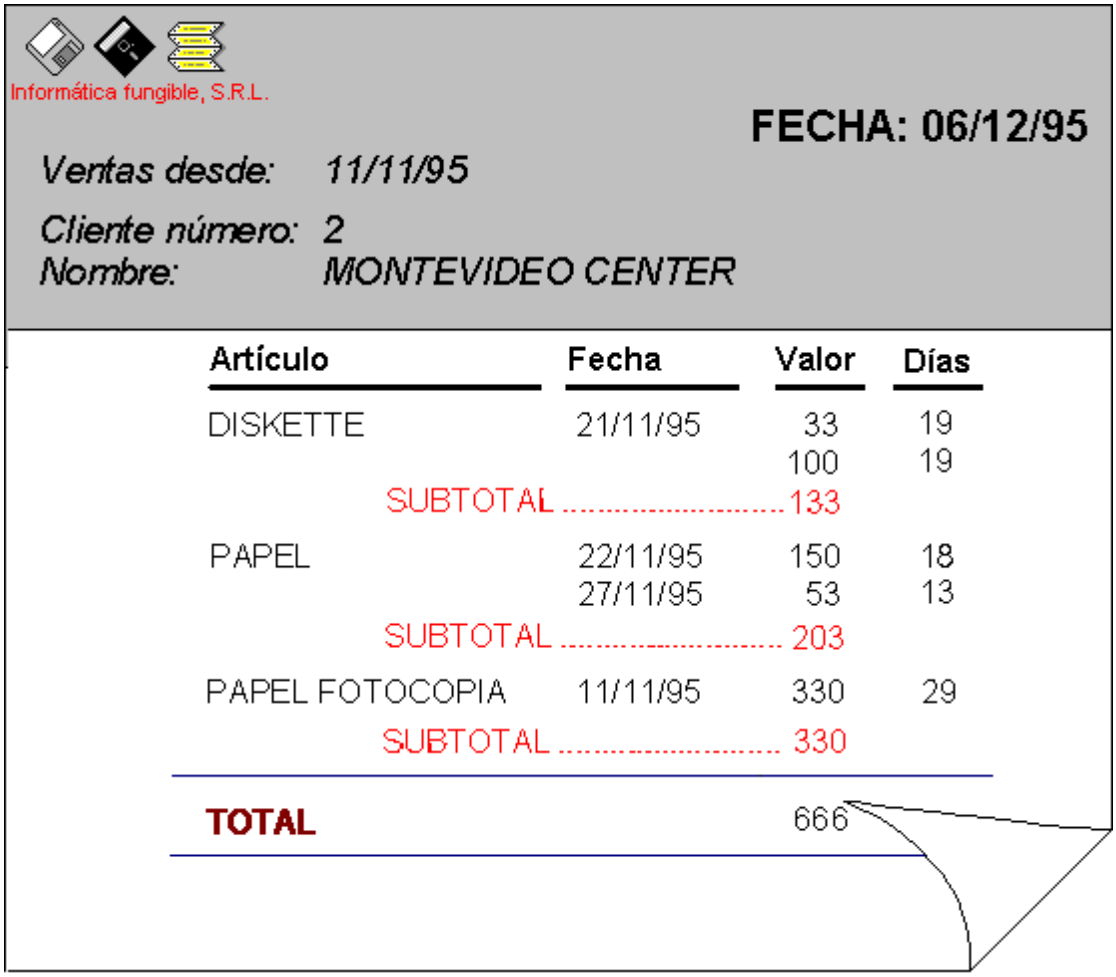

Figura 3: Ejemplo de reporte con subtotales

Hasta aquí los conceptos básicos del diseño de reportes en Oracle con *SQLReport*. En este caso, como en todos los artículos anteriores, mi recomendación es que a partir de lo estudiado el lector puede seguir profundizando en el tema con el uso de los manuales de Oracle, que son didácticos y con muchos ejemplos ilustrativos.

En nuestro próximo artículo estudiaremos el tema del diseño de menús y la integración de los diferentes módulos estudiados en nuestra serie de artículos.

ácle básico (y VI): SQLMenu

SQLMenu es el producto de *ORACLE* destinado a producir árboles de menúes que permiten al usuario desplazarse fácilmente a través de su aplicación, facilitando a su vez, la integración con los diferentes módulos de *ORACLE* analizados en nuestros artículos anteriores permitiendo, de esta forma, la conexión con *SQLForms*, *SQLReport*, *SQLPlus*, *PL/SQL* y tareas del sistema operativo.

La integración de módulos, herramienta que permite desde un producto de Oracle invocar otros, también está presente en *SQLForms*, desde donde, como veremos en este artículo, se pueden realizar tareas del sistema operativo con llamadas a los módulos SQLReport y SQLPlus.

Definiciones

Comenzaremos explicando las diferentes partes que componen el diseño de un menú en *SQLMenu*. Ellos son:

Aplicación — Conjunto de uno o más menúes interconectados para realizar las tareas necesarias del sistema y para el cual se definen su nombre, nombre del archivo ejecutable, fecha de creación, nombre del usuario, número de la versión, fecha de la última modificación, directorio de ubicación y su identificación.

Menú — Lista de opciones o items que realizan las tareas específicas del sistema. Cuando se crea la aplicación el primer menú que se debe definir es el principal, cuyo nombre tiene que ser el mismo que el de la aplicación. Este será el menú que se activará en forma automática al cargar la aplicación.

Items — Define las opciones del menú y su correspondiente acción. Un ítem puede llamar a otro menú, ejecutar un comando o un módulo de Oracle, así como cualquier otro programa. Para cada ítem se define su posición en el menú, tipo de comando, permisos de los usuarios, texto de identificación y la línea del comando a realizar.

Parámetros — Variable que se carga en el momento de ejecución del menú y para la cual se definen un nombre de dos letras, cantidad máxima de caracteres, texto en pantalla en el momento de su ingreso, si es o no obligatorio, con o sin conversión a mayúsculas y su valor por defecto.

Existen 5 parámetros del sistema cuya información podemos verla en la tabla 1.

Parámetro	Valor
UN	Nombre del usuario.
PW	Contraseña del usuario.

AD	Directorio actual.
SO	Opción seleccionada del menú.
TT	Tipo de terminal en uso.

Tabla 1. Parámetros del sistema.

Los parámetros se definen en la línea de comandos precedidos por *&*, como por ejemplo *&UN*. Para los bloques de código en *PL/SQL* se hace referencia precedido con dos puntos de la forma *:UN*.

Tipos de comandos

En *SQLMenu*, desde un menú, se pueden realizar varios tipos de comandos. Pueden verse en la tabla 2.

Tipo	Descripción
1	Llamada a un submenú.
2,3	Tarea del sistema operativo sin y con pausa.
4	Conexión con SQLForms.
5	Conexión con SQLPlus.
6	Realiza un Macro de SQLMenu.
7	Realiza un procedimiento en PL/SQL.

Tabla 2. Lista de comandos que se pueden realizar

A continuación, para conocer cada tipo de comando, diseñaremos una aplicación llamada *CLIENTE*, cuyo menú principal, llamado también *CLIENTE*, esta compuesto de 6 submenúes (tipo de comando 1), como se muestra en la tabla 3.

Menú	Nº Item	Tipo comando	Línea comando
CLIENTE	1	1	FORMA
	2	1	REPORTE
	3	1	LISTADO
	4	1	PROCEDIMIENTO
	5	1	MACRO
	6	1	SISTEMA

Tabla 3. Submenúes del menú *CLIENTE*

La pantalla de nuestro menú *CLIENTE* se presenta de la siguiente forma:

```
FORMA  REPORTE  LISTADO  PROCEDIMIENTO  MACRO  SISTEMA
_____________________ ___CLIENTE_____________________ ___
```

Pasaremos a detallar cada submenú, para de esta forma detenernos en las particularidades de cada tipo de comando.

FORMA Realiza tareas del comando tipo 4 para la integración de SQLMenu con SQLForms. En este caso pasamos como línea de

comando la orden RUNFORM con la información necesaria para la conexión, como son, el nombre del usuario, su contraseña y el nombre de la forma a activar. La sintaxis de este línea de comando corresponde a la de la orden RUNFORM desde el sistema operativo, siendo necesario, en este caso, pasar el nombre del usuario y su contraseña, definidos al momento de realizar la conexión con SQLMenu, a través de los parámetros UN y PW.

En la tabla 4 se muestra como se activa la forma *CLIENTES* desde el menú *FORMA*.

Menú	Nº	Tipo de comando	Línea de comando
FORMA	1	4	RUNFORM CLIENTES &UN/&PW

Tabla 4. Cómo activar la pantalla *CLIENTES*

REPORTE

La conexión con *SQLReport* no es de forma directa como lo es con *SQLForms*, sino que se debe realizar a través del tipo de comando 2, que se encarga de pasar la línea de comando del menú al sistema operativo para que lo procese. En este caso se siguen las mismas reglas detalladas anteriormente para *SQLForms*. En la tabla 5 se muestra como se activa el reporte llamado *VENTAS*.

Menú	Nº	Tipo de comando	Línea de comando
REPORTE	1	2	RUNREP VENTAS &UN/&PW

Tabla 5. Cómo activar el reporte *VENTAS*

LISTADO

Los listados diseñados en *SQLPlus* se activan con el tipo de comando 5, que es el que permite la integración de *SQLMenu* con *SQLPlus*. La identificación del usuario y su contraseña se logra con los parámetros *UN* y *PW* como explicamos en los casos anteriores.

Menú	Nº Item	Tipo de comando	Línea de comando
LISTADO	1	5	SQLPlus -S &UN/&PW @CLIENTE
	2	5	SQLPlus &UN/&PW \FAC\VENTAS

Tabla 6. Cómo activar el listado *LISTADO*

Los archivos de listado, por ejemplo *CLIENTE.sql*, debe tener *EXIT* como último comando, con el fin de garantizar la desconexión de *SQLPlus* y el regreso a *SQLMenu*. El parámetro *-S* de *SQLPlus* suprime todas las visualizaciones de especificaciones del *SQLPlus* por pantalla.

PROCEDIMIENTO Para ejecutar un procedimiento del sistema o del usuario, así como un bloque anónimo de *PL/SQL* escrito directamente en la línea de comando del menú, se utiliza el tipo de comando 7.

Menú	Nº	Tipo de comando	Línea de comando
PROCEDIMIENTO	1	7	EXIT_MENU;
	2	7	PROCEDURE CAMBIO IS
			BEGIN
			...
			END;
	3	7	NUEVO;

Tabla 7. Cómo ejecutar un procedimiento.

En nuestro ejemplo, el ítem 1 es un procedimiento del sistema, el 2 es un bloque anónimo y el 3 es un procedimiento creado por el usuario.

En el fuente 1 vemos la definición del procedimiento *NUEVO*, para de esta forma explicar algunos de ellos:

```
Fuente 1: Procedure Name: NUEVO
------Procedure Text -------------
PROCEDURE NUEVO IS
BEGIN
      NEW_USER('EIDOS/ESPAÑA');
      NEW_APPLICATION('VENTAS');
      MAIN_MENU;
      OS_COMMAND('DIR');
      EXIT_MENU('Fin del Menú');
END;
```

Donde:

NEW_USER Desconecta al usuario actual y conecta un nuevo usuario.
NEW_APPLICATION Cambia de aplicación.
MAIN_MENU Pasa al menú principal.
OS_COMMAND Realiza un comando del sistema operativo.
EXIT_MENU Salida del Menú.

MACRO Existe un conjunto de macros incorporadas al *SQLMenu* que pueden ser llamados desde un menú, como los que se muestran la tabla 8.

Menú	Nº	Tipo de comando	Línea de comando

			o
MACRO	1	6	NEWUSER;
	2	6	NEWAPL;
	3	6	MAINMENU;ASSIGN DK=A:;

Tabla 8. Ejecución de macros que pueden ser llamadas.

Los macros tienen un procedimiento equivalente en *SQLMenu* y viceversa. En los manuales de Oracle existe una tabla de correspondencia entre *Macro-Procedimiento-Tecla de Función*. Esta dualidad está dada con el fin de mantener la compatibilidad con versiones anteriores de *SQLMenu*.

Algunas macros tienen argumentos, como la macro *ASSIGN*, cuyo objetivo es asignar un nuevo valor a un parámetro definido por el usuario. La definición y el uso de los parámetros lo veremos más adelante.

SISTEMA

Los comandos de tipo 2 y 3 son los que se usan para hacer tareas desde el sistema operativo, pasando una línea de comando directamente al sistema operativo para su ejecución, como ya vimos anteriormente para el caso de los reportes. La diferencia consiste en que después de realizar el comando de tipo 2 se retorna al menú inmediatamente, y con el tipo 3 se realiza una pausa antes de retornar al menú, quedando a la espera de que el usuario presione alguna tecla para continuar.

Menú	Nº	Tipo de comando	Línea de comando
SISTEMA	1	2	CHKDSK &DK
	2	3	DIR &DK

Tabla 9. Ejecución de tareas del sistema

En este ejemplo se usa el parámetro *DK* para identificar la unidad de disco con la cual se va a trabajar. En la figura 1 se muestra la pantalla de definición de este parámetro.

	Parameter Definition	
Parameter	DK	[x] Echo
Size	2	[ ] Fixed Length
Prompt	Diskette A: B: o disco (Select Menus)	[ ] Required [ ] Upper Case
Default Hint	C:	

Figura 1. Pantalla de definición del parámetro *DK*.

Una vez seleccionado la opción del menú *SISTEMA* que hace referencia al parámetro *DK*, aparecerá en pantalla, para ingresar el valor del parámetro, el siguiente cuadro:

```
Enter Parameter Value(s)
Diskette A: B: o disco        C:
```

Integración desde SQLForms

La integración de módulos, como los casos que acabamos de estudiar, también esta presente en *SQLForms*, desde donde podemos activar los módulos *SQLReport* y *SQLPlus* a través del sistema operativo.

En artículos anteriores estudiamos por separado *SQLForms* y *SQLReport*, ahora pasaremos a ver como se realiza la integración de estos dos módulos. Las definiciones a realizar para esta integración son las siguientes:

En *SQLReport*:

- Parámetros y sus valores por defecto. Esta es la vía que tenemos para pasar los datos necesarios, desde SQLForms, en el momento de la ejecución del reporte.

En *SQLForms*:

- Campos de la forma donde se van a ingresar los valores a pasar como parámetros al reporte.

- Procedimiento que invoca a *SQLReport* con sus correspondientes parámetros. Para esto se usa la orden *HOST*, procedimiento del sistema que tiene como función enviar un comando al sistema operativo, en este caso *RUNREP*, garantizando el paso de variables a través de los parámetros y el retorno automático a *SQLForms*.

Veamos primero la definición de los parámetros en *SQLReport*, cuyo objetivo es controlar diferentes aspectos del reporte, como son: dispositivo de salida, tipo de impresora, número de copias y datos específicos del usuario. Para ello retomaremos el ejemplo *VENTAS*, diseñado en nuestro artículo anterior *ORACLE BASICO V* (ver ***Algoritmo*** Nº13): *Reportes con SQLReport*. En la tabla 10 podemos ver los parámetros del sistema y del usuario con sus respectivos significados.

	Nombre	**Tipo**	**Ancho**	**Valor**	**Identificación**
SISTEMA	DESTYPE	CHAR	80	Printer	Tipo de salida: Screen, File o Printer
	DESNAME	CHAR	80	PRN	Nombre de Dispositivo o archivo
	DESFORMAT	CHAR	80	dflt	Especifica tipo de formato de la impresora para sus códigos de control.
	COPIES	NUM	2	1	Número de Copias

USUARIO	XCOD	NUM	10	2	Código Cliente
	XDIAS	NUM	5	30	Cantidad de Días

Tabla 10. Parámetros del sistema y del usuario.

Ahora crearemos, en *SQLForms*, el procedimiento llamado *REPORTE*, cuyo objetivo es realizar el reporte *VENTAS*. (Véase el fuente 2)

```
/* Fuente 2 */
PROCEDURE REPORTE IS
 BEGIN
       DECLARE
                          /*
                          Variables locales de memoria, de tipo carácter,
cuyo
                          objetivo es cargar los valores de los campos
numéricos
                          de la forma.
                          */
             XC CHAR(10);
              XD CHAR(5);
             BEGIN
                          /*
                          Asignación de los valores de los campos de la
forma
                          YCOD y YDIAS a las variables locales con su
correspondiente
                          conversión a carácter.
                          */
              XC:=TO_CHAR(:YCOD);
              XD:=TO_CHAR(:YDIAS);
                          /*
                          Llamada a SQLReport por medio de la orden HOST,
                          con la correspondiente cadena de caracteres
                          de definición de parámetros.
                          */
                   HOST ('RUNREP REPORT=VENTAS
                                 USERID=EIDOS/ESPAÑA
                                 PARAMFORM=NO
                                 DESTYPE=PRINTER
                                 DESNAME=PRN
                                 COPIES=3

                                 XCOD='||XC||
                                 ' XDIAS='||XD
                          );
      END;
 END;
```

Donde:

REPORT=VENTAS Indica que el reporte a ejecutar esta en el archivo *VENTAS.rep*, donde quedó registrado el resultado de la compilación efectuada desde *SQLReport*.

USERID=EIDOS/ESPAÑA Identifica el nombre del usuario y su contraseña.

PARAMFORM=NO Indica que los parámetros se pasan directamente en la línea de comando, de lo contrario se solicitaría su ingreso por pantalla en el

momento de la ejecución.

DESTYPE=PRINTER Salida por impresora.

DESNAME=PRN Nombre de la impresora.

COPIES=3 Cantidad de copias.

En esta integración no existe límite en la cantidad de parámetros a transferir, existiendo la posibilidad de construir un archivo de parámetros. Por ejemplo, si escribimos los parámetros en el archivo *ventas.para*, la conexión con *SQLReport* desde el procedimiento *REPORTE* sería de la siguiente forma:

```
      HOST   ('RUNREP REPORT=VENTAS
                     CMDFILE=VENTAS.PAR'
             );
```

La integración de *SQLForms* con *SQLPlus* cumple las mismas reglas detalladas anteriormente para *SQLReport*, sólo varía la sintaxis en la orden *HOST* ya que ésta corresponde a las especificaciones del comando *SQLPLUS* como se muestra a continuación:

```
HOST   ('SQLPLUS EIDOS/ESPAÑA @LISTADO '||
             :F1||
             ' '||
             :F2
      );
```

Donde:

EIDOS/ESPAÑA Identificación de usuario y su contraseña.

@LISTADO Archivo que contiene los comandos SQL a ejecutar.

:F1 y :F2 Campos de la forma, de tipo carácter, cuyos valores se pasa como parámetros al *SQLPlus* en el momento de la ejecución.

Con este artículo llegamos al final de la serie sobre *Oracle Básico*, donde hemos analizado sus aspectos básicos, así como sus rasgos más significativos. Espero haber logrado, por un lado, introducir al lector en el mundo de bases de datos relacionales y *SQL* y, en el caso de aquellos que ya desarrollan aplicaciones en este entorno, impulsarlos a seguir avanzando en el estudio de Oracle.

Bibliografía

Oracle 7 Manual de Referencia
Koch, George.
Osborne/McGraw-Hill
1994

Oracle Manual de Referencia.
Koch, George.
Osborne/McGraw-Hill.
1992

Mastering Oracle.
Cronin, Daniel.
Hayden Books.
1990